Exhibit P. Registries and Repositories – XML/SGML Name Registration (about Nov. 1998) accessed at http://xml.coverpages.org/registryColl.html on October 26, 2006, 30 pp.

Registries and Repositories - XML/SGML Name Registration

[CR: 20030113]

[See also the slightly more maintained document XML Registry and Repository.]

The development of new vocabularies and the design of "namespace" syntaxes have increased public interest in registration authorities and authentication services which could be set up to manage name conflicts. Facilities are needed for support of globally-unique names, persistent links and resources, name (public identifier) resolution, mapping between public and system identifiers, etc. Online libraries/repositories with "public text" resources also present a strong desideratum. Several initiatives for registries and repositories have been announced. A few of the early initiatives which have been publicized are cited below, along with pointers to the SGML resources.

Note/Caution: This document contains information cribbed from all over the Net and thrown together into a crude outline. Just 'unedited notes'. Initially, it served as the basis for a report delivered to the OASIS Technical Committee, Fall 1998. Anyone who reads it might get a reasonable overview of what people were thinking about in terms of [XML] "registries" in about November 1998. This is not a maintained document. It is posted unofficially for use by the OASIS Registry and Repository Technical Committee.

Contents

NIST

R-745

NIST ESTABLISHES COLLABORATION XML REGISTRY

The NIST Identifier Collaboration Service (NICS) project, funded through the Advanced Technology Program, has established an experimental collaborative registry for XML (Extensible Markup Language). XML is designed to enable the use of SGML on the World Wide Web. An XML DTD is the formal definition of a particular type of XML document. The DTD sets out what names can be used for elements, where they may occur, and how they all fit together. Before now, researchers and vendors have faced conflicts from similarly named XMLs and element names. Vendors can now publicly register their names and their download locations, thus avoiding future conflicts while publicizing existence and availability. This project is conducted in the Manufacturing Collaboration Technologies Group of the Manufacturing Systems Integration Division of the Manufacturing Engineering Laboratory.
For further information, contact Don Libes, (301) 975-3535

- "NIST Establishes Collaboration XML Registry." 5/1/98 [local archive copy]

XML/EDI - XML Repository Working Group

"Repositories provide a standard reference point for understanding data and processes. In the context of XML and EDI repositories provide the means to implement strategic business systems. Repositories also provide the link between business process designs (UML) and their physical implementation (XML). The goal of the [XML Repository Working] Group is to create a proposed standard on global XML repositories for submittal to the World Wide Web Consortium (W3C), Object Management Group (OMG), and UN/EDIFACT."

"The combination of XML and EDI semantic foundations, called XML/EDI, will provide a complete framework where a set of different technologies work together to create a format that is usable by applications as well as humans. These technologies are XML, EDI, Templates, Agents and Repository. . . The repository is a location where shared Internet directories are stored and where users can manually or automatically look up the meaning and definition of XML/EDI Tags. The repository is in fact the semantic foundation for the business transactions."

WHAT ARE XML REPOSITORIES?
The combination of XML and EDI semantic foundations, called XML/EDI, will provide a complete framework where a set of different technologies work together to create a format that is usable by applications as well as humans. These technologies are XML, EDI, Templates, Agents and Repository. These components all work together to create an XML framework for business use:

- The XML tags can actually replace existing EDI segments or data-element identifiers, which produces a somewhat bigger file than an EDI file (before compression), but in which all the labels of the data-elements (in other words the descriptions or explanations) can optionally be used as XML tag names. The XML DTD (Document Type Definition) that works with the XML carries the structural information and attribute information from the original EDI transaction model.
- The repository is a location where shared Internet directories are stored and where users can manually or automatically look up the meaning and definition of XML/EDI Tags. The repository is in fact the semantic foundation for the business transactions.
- The templates are essentially rules that determine how the XML files should be interpreted. It can define the layout of the file and is supplemented by DTDs that enable transaction operability. Templates can be stored in repositories for global reference.
- The agents can interpret the templates to perform whatever job needs to be done (such as accessing local data stores), but they can also interact with the transaction and help the user to create new templates for each specific task.

R-746

What makes XML/EDI different from previous initiatives, is that it can use the know-how of business processes captured in EDI messages. This is then delivered via the Internet and in a Web environment. Thereby the same file can be viewed by a user in a desktop tool, or can be processed by an application component on a server. One core concept this enables is that while traditional EDI is "door to door" between business partners, XML/EDI can flow in through the "door" and be used in multiple locations within organizations.

- Home Page
- XML/EDI Repository Focus
- Repositories Position Document & Presentation on UDEF and XML/EDI
- [August 03, 1998] "Business to Get XML Repository." By Jeff Walsh and Matthew Nelson. In *InfoWorld* Volume 20, Issue 31 (August 3, 1998). Summary: "The XML/EDI Group has announced it is developing a repository for business transactions based on the Extensible Markup Language (XML). . ."
- Prototype Demonstration Project: X.12 Based Data Exchange Scenarios using XML and Repository Technology. By René Kasan. XML/EDI Repository Group. EXtensible Markup Language (XML) is a fast emerging technology whose potential is recognized by many companies and organizations. One particular advantage of XML is its ability to describe the content of almost any digital document. Therefore, many believe that XML might prove useful for doing business electronically on the Internet. The Center for the Study of Electronic Commerce in the Daniels College of Business at the University of Denver formed a research team (DCB Team) to prototype an XML/EDI operating environment in order to explore its feasibility. The team is comprised of two graduate students, René Kasan and Leonardo Cabrerra, one faculty member, Don McCubbrey, and two practitioners, Ron Schuldt and Will Thayer. Most of the work is being performed by the students with oversight and counsel from the others. The DCB Team also has ties to the XML/EDI group (www.xmledi.com) and the XML/EDI Repository Group (www.xmledi.com/repository), whose members will be kept informed on our work and encouraged to participate as our project moves forward. This document describes a practical prototype demonstration project for different EDI scenarios using XML."

EEMA EDI/EC Work Group - XML/EDI

EEMA EDI/EC Work Group Proposal. "The EEMA EDI Work Group would like to propose to CEFACT the establishment of a Global Repository for the translation of XML tags in UN/EDIFACT and human language on the Internet. The EEMA EDI Working Group is prepared to assist in the set up and operation of such a repository, which could be crucial in the advancement of the use of EDI over the Internet. . . . When the fusion between ANSI X12, EDIFACT and all other EDI standards takes place in a proper way it should be under the auspices of the UN so it is global, public domain, open and available for anyone. Today this is not the case with the ANSI standards and many other EDI standards that are only available at considerable cost. Of course today the EDIFACT standard is already in the public domain and can easily be obtained through the Internet."

Referrences:

- Proposal for a UN Repository for XML Tags Based on UN/EDIFACT CEFACT. TRADE/CEFACT/1998/CRP. 25. 31 August 1998. SOURCE: Dick Raman, Head of Delegation, EEMA. "This paper proposes that CEFACT establish a Global Repository for the translation of UN/EDIFACT in XML tags. The European Electronic Messaging Association (EEMA) EDI Working Group is prepared to assist the UN/ECE Secretariat in the set up and operation of such a repository, which could be crucial in the advancement of the use of EDI over the Internet." [local archive copy]
- Presentation on XML-EDI by Dick Raman - 19 June 1998 - CEFACT-SIMAC - Geneva. By Dick Raman
- Proposal for a UN Repository for XML/EDI [local archive copy]

R-747

XML Exchange

"Welcome to CommerceNet's XML Exchange, the forum for creating and sharing document type definitions. This isn't an online magazine, and it isn't some glass tower where elite experts publish their thoughts. CommerceNet's XML Exchange is a public place for anyone to ask questions and discuss their challenges and successes in making XML work."

- XML Exchange originally created by XMLSolutions, LLC.
- XML Exchange Subject Areas

```
XML Exchange Categories: [1998]
Automotive
Business productivity
Chemical
EDI
Education
Financial
Genealogy
General
Geophysics
Government
Healthcare
History
Human resources
Insurance
Library
Manufacturing
Mathematics
Metadata
Military
Real estate
Web Management
Workflow
```

CommerceNet

- "CommerceNet Acquires XML Exchange From XMLSolutions. Foundation for Global eRegistry Service Vital to XML-based Electronic Commerce." - "Only ten weeks after its initial launch, the XML Exchange (http://www.xmlx.com) has been acquired [by CommerceNet]." CommerceNet's XML Exchange will be 'a component of its eRegistry Service'. [local archive copy]
- "CommerceNet to Use XML Web Site for Registry." By Nancy Weil. In *InfoWorld Electric* (July 16, 1998), Posted at 9:30 AM PT. "A Web site dedicated to Extensible Markup Language (XML) has been acquired by a global Internet-commerce consortium that intends to use the site as the core of a registry service to ensure interoperability and information exchange among developers. XML Exchange (http://www.xmlx.com), which was launched 10 weeks ago, will now be run by CommerceNet, a non-profit I-commerce industry group with more than 500 worldwide members, CommerceNet announced. The Web site was acquired for an undisclosed sum from XMLSolutions, a Washington consulting company." [local archive copy]
- [May 26, 1998] "CommerceNet To Offer an eCommerce Registry Service. Service Will Advance the Future

R-748

Development and Growth of XML-based Electronic Commerce." Based in part upon funding from NIST. [local archive copy]

- [May 26, 1998] Press release, May 21, 1998. "CommerceNet To Offer an eCommerce Registry Service. Service Will Advance the Future Development and Growth of XML-based Electronic Commerce." Based in part upon funding from NIST. [local archive copy]
- [January 19, 1999] "Commerce One Acquires Veo Systems, Inc. Leading Electronic Commerce Solutions Provider Acquires XML Technology Leader to Rapidly Expand Open Internet Trading Communities." - "Commerce One, Inc., the global leader in enterprise procurement solutions, today announced that it has acquired privately held Veo Systems, Inc., the leading solutions provider of Extensible Markup Language (XML)-based open commerce networks. With this acquisition, Commerce One will rapidly accelerate the pace of development of industry-leading XML-based business-to-business electronic commerce solutions to enable open Internet trading communities." See also the anouncement on the Commerce One Web site. [local archive copy]

---

eCo Framework Project - Working Group

In response to the growing use of the eXtensible Markup Language (XML) and proliferation of independent XML-based eCommerce protocols, CommerceNet is chartering the eCo Framework Project and Working Group.

The eCo Framework Project is chartered by CommerceNet with sponsorship by Veo Systems Inc. and other companies to be added. The goal of the project is to develop a common framework for interoperability among XML-based application standards and key electronic commerce environments. The project's working group will develop a specification for content names and definitions in electronic commerce documents, and an interoperable transaction framework specification.

The eCo Framework Working Group is chartered to define a common framework from an ever-growing complement of electronic commerce related specifications, including Catalog Information Specification, Channel Definition Format (CDF), Common Business Library (CBL), Electronic Data Interchange (EDI), Internet Content Exchange (ICE), Open Buying on the Internet (OBI), Open Financial Exchange (OFX), Open Trading Protocol (OTP), and XML. The working group, modelled after the successful Davenport and XML working groups, includes industry experts from 3Com, American Power Conversion, Compaq/Tandem, Harbinger, Hewlett-Packard, IBM, Intel, Microsoft, NEC, Netscape, NTT, Sun Microsystems, RosettaNet, and Veo Systems.

- ECo Home
- eRegistry submission process: Define a process for review and incorporation of submitted XML DTDs into an eCo eRegistry. The eCo Framework Working Group is not responsible for implementation or ongoing use of the process. This work product will be known as the "eCo eRegistry Submission Process".
- Requirement 10: Define registries in a manner which can describe the syntax, semantics, source, destination and mapping. Recommend a process by which existing XML languages can be placed into the eCo architecture and evaluated. The requirements and criteria against which implementations/protocols can be evaluated should be clearly established.
- eCo Framework Project: Team

Veo Systems Common Business Language (CBL)

CBL (Common Business Library) is an extensible, public collection of DTDs and modules that companies can customize and assemble to develop XML-based commerce applications. Because it re-uses common semantic components, CBL helps speed the development of e-commerce standards and applications-and, more importantly, facilitates their interoperation. Toward this

end, Veo is sponsoring a project through CommerceNet's eCo Working Group to create a public XML registry.

- Veo Systems Home Page
- Common Business Library Frequently Asked Questions

Address:
Veo Systems, Inc.
2440 West El Camino Real, Seventh Floor
Mountain View, CA 94040
Phone: 650.938.8400
Email:
general: info@veosystems.com
sales:sales@veosystems.com
employment: jobs@veosystems.com

Content-X

Connected to ICE: "Content-X.com - The Web's ICE information source Content-X is a news and discussion site for web publishers interested in Vignette Corporation's Information & Content Exchange (ICE) protocol. ICE is an exciting new standard designed to automate the exchange of digital assets between businesses. Content-X provides a community forum for exploring the uses of ICE, and its effect on the content syndication industry. ICE is XML-based, so we also run features on general XML topics. The ICE 1.0 specification is now a W3C NOTE."

Has a 'Syntax Repository'

On November 12, 1998, the DTDLIST.HTML document was:

```
"Repository

ICE DTD (ICE.dtd), for news documents. To read the entire ICE specification, click
here

NITF DTD (nitf-x.dtd), for news documents. To read about it, click here

FlixML DTD For more information about FlixML see our Technology page.

WDDX DTD Read about what WDDX is. " - viz, 4 dtd links
```

Syntax Repository from Content-X.com. "XML allows the creation of domain-specific markup languages. Markup languages will be created for different segments of the content syndication business, making the automated exchange of content even more efficient. Content-X will be creating a repository for these DTDs, schemas, and logic elements. All DTDs will be located at: **http://www.content-x.com/repository/dtdname.dtd**. Applications will be able to pull DTDs from this URL as needed."

http://www.eontent-x.com/syntax/dtdlist.html

OASIS - Organization for the Advancement of Structured Information Standards

R-750

- The OASIS Technical Committee is considering development of an SGML/XML registry and repository. This effort is being led by Terry Allen, of Veo Systems, Inc. [Commerce One].
- See information on the SGML Open (OASIS) 'CATALOG' and identifiers in the dedicated database entry: Catalogs, Formal Public Identifiers, Formal System Identifiers, or in the Entity Management Resolution.
- [December 08, 1998] "Managing Names and Ontologies: An XML Registry and Repository." By Robin Cover. From Sun Microsystems, Technology & Research. November, 1998. "Early adoption of XML by a host of industry partners is thus creating a wealth of opportunity for information reuse and collaborative distributed network computing over the Web. At the same time, the rapid emergence of XML DTDs and vocabularies from industry and government sectors has focused public attention upon issues of resource identification, classification, cataloging, and delivery that hinder reuse and interoperability. The results of new collaborative endeavors are not necessarily easy to identify and access on the Internet. Simply put: XML resources are not nearly as discoverable and reusable as they deserve to be." [local arehive eopy]

ISO 11179

ISO 11179 might be considered 'for a technical and conceptual framework' in the OASIS RegRep TC work (TA). Note Registration of data elements (part 6) and 11179-2: Classification for Data Elements.

ISO 11179 with X3.285. DTD Element Index. DTD work from ISO 11179 and X3.285 by Terry Allen (Veo Systems); HTML presentation of ISO 11179 with X3.285 facilitated by Norm Walsh's DTDParse. **NB:** this is a transient URL for the ISO 11179 DTD, so please **do not** create public bookmarks to it.

Development of ISO/IEC 11179 - Specification and Standardization of Data Elements

From: http://guagua.echo.lu/oii/en/meta.html#ISO11179

ISO 11179 - *Expanded name*
Specification and Standardization of Data Elements

Area covered
Standard for describing data elements used in databases and documents.

Sponsoring body and standard details

- ISO/IEC JCT1/SC32
- ISO/IEC 11179 *Information technology -- Specification and standardization of data elements*
    - Part 1: *Framework for the specification and standardization of data elements*
    - Part 2: *Classification for Data Elements*
    - Part 3: *Basic Attributes of Data Elements*
    - Part 4: *Rules and Guidelines for the Formulation of Data Definitions*
    - Part 5: *Naming and Identification Principles for Data Elements*
    - Part 6: *Registration of Data Elements*

Characteristics/description
ISO 11179 specifies basic aspects of data element composition, including metadata. The standard applies to the formulation of R-751

data element representations and meaning as shared among people and machines; it does not apply to the physical representation of data as bits and bytes at the machine level.

An ISO 11179 data element is composed of three parts:

- an *object class* is a set of ideas, abstractions, or things in the real world that can be identified with explicit boundaries and meaning, and whose properties and behavior follow the same rules;
- a *property* is a peculiarity common to all members of an object class;
- a *representation* describes how the data are represented, i.e. the combination of a value domain, datatype, and, if necessary, a unit of measure or a character set.

The combination of an object class and a property is called a *data element concept* (DEC). A *value domain* is a set of permissible (or valid) values for a data element.

Part 2 of ISO/IEC 11179 provides procedures and techniques for associating data element concepts and data elements with classification schemes for object classes, properties and representations.

Part 3 specifies attributes of data elements. It is limited to a set of basic attributes for data elements, independent of their usage in application systems, databases, data interchange messages, etc. Part 4 provides guidance on how to develop unambiguous data element definitions.

Part 5 provides guidance for the identification of data elements, including the assignment of numerical identifiers that have no inherent meanings to humans, icons (graphic symbols to which meaning has been assigned), and names with embedded meaning. Part 6 provides instruction on how a registration applicant may register a data element with a central Registration Authority.

Usage (Market segment and penetration)
Parts 3-6 were published by ISO between 1994 and 1997, but the other parts are only available as working documents awaiting approval by ISO.

Further details available from:
ISO or local national standards body.

Metadata and Data Registries - and ISO/IEC JTC1/SC32/WG2 NWI

Some documents on the work of ISO/IEC JTC1 SC32 WG2 ('Metadata Registries') are available from NIST:

- XML Representation for Data Registry Metadata Exchange - New Work Item Request for ISO/IEC JTC1/SC32/WG2. [local archive copy]
- L8 Subcommittee on XML New Work Item [local archive copy]
- XML extensions for ISO/IEC 11179- Work Plan As of October 23, 1998. [local archive copy]
- Website for the Open Forum on Metadata Registries, [local archive copy]

ISO/IEC Open Forum on Metadata Registries. The Open Forum is Organized by: International Organization for Standardization / International Electrotechnical Commission (ISO/IEC) Joint Technical Committee 1 (JTC 1), Subcommittee 32 - Data Management and Interchange (SC32), Working Group 2 - Metadata (WG2). "The Open Forum will be held February 16-19, 1999 in Washington DC. You are invited to register to attend. You may also participate by contributing relevant papers or web links. Participants from private enterprise, government, academia and standards organizations will

discuss the development and operation of metadata registries, particularly those based on ISO/IEC 11179. Practitioners and standards developers will discuss progress in efforts to manage the content (semantics) of data that is shared within and between organizations or disseminated via the World Wide Web. Presentations and discussions will cover tutorials, implementation plans/experiences, proposed new work items for standards,and related topics."

Panel 4 - Using XML to Embed and Exchange 11179 Metadata: Panel organizers - Frank Olken and John McCarthy Lawrence Berkeley National Laboratory

The Data Registry Community intends to develop a set of XML tags for 11179 metadata, just as other professional communities have developed tagsets such as MathML and ChemML. This work will help make ISO/IEC 11179 a useful part of new XML-enabled technologies. One goal of this effort is to facilitate interoperation among metadata registries. Another goal is to enable XML applications (such as XML-EDI) to directly access metadata registries using a standard syntax and semantics. This panel brings together experts from the ISO and NCITS committees responsible for 11179, World Wide Web Consortium (W3C) committees working on XML extensions, Electronic Data Interchange committees, and the Object Management Group (OMG) to discuss the opportunities and challenges of this 11179 XML tag set development effort."

Metadata and Data Registries - Seminar

"Databases on the Web: The Role of Metadata and Data Registries." Frank Olken and John McCarthy, Lawrence Berkeley National Laboratory. SIM Information Access Seminar. XML (Extensible Markup Language) offers the promise of providing readily parsed structure for web-based information. It is widely expected that it will have a major impact on data exchange formats and web access to databases. . . We will discuss ongoing work in W3C and elsewhere to provide schema languages for XML (and a related metadata language RDF (Resource Description Framework). Our discussion will include a brief description of the data models embodied in XML, XML+Xlinks, and RDF. We will then discuss various recent proposals: RDF Schema, XML Data, DCD (Data Content Definition), OMG's XMI (Extensible Metadata Interchange), and our own work on XDT (Extensible Data Types). This work on schema specification languages is intended for use by metadata content standardization organizations such as FGDC (Federal Geospatial Data Committee) and the Dublin Core Group. We will not discuss content standards in our talk. We will also discuss related work in NCITS L8 (the U.S. TAG for ISO SC32) (formerly ANSI X3L8) on data registries, e.g, the standards ISO 11179 and ANSI X3.285. Related work in NCITS T2 (formerly X3T2) on ontology standardization will be briefly mentioned."

Ontology.org

Initiatives are already underway to agree standard XML DTDs within specific vertical industries or markets. Many industries, for example, electronics, automotive and aerospace, have already established sophisticated SGML DTDs. There will be considerable interest in converting these to XML for use in practical Internet commerce applications. The recent formation of organisations such as RosettaNet, X-ACT (OASIS) and grass roots communities such as XML Exchange, XML.com and Xmlu.com indicate the depth of interest in the application of XML. CommerceNet, the leading worldwide electronic commerce consortium, have recently announced that they are developing an eCommerce Registry service, which will accelerate the adoption of XML-based electronic commerce. The process will allow anyone to submit XML DTDs with immediate availability to anyone who wants access.

- Ontology.Org Home Page
- Ontology.Org FAQ
- Press release October 9, 198. Ontology.Org and CommerceNet Form Strategic Alliance, Success of XML for eCommerce Accelerated by Advanced Knowledge Representation Techniques :

R-753

Brown University Scholarly Technology Group (STG)

```
> If you have a 3-4 sentence para, or 2-4 item list saying what
> you have implemented in prototype/alpha for the "automated"
> FPI registration facility . . .
> [ca November 1998]
```

```
STG's goal is to place online, relatively quickly, an XML-
validation/FPI-registration system - and to do it from some
neutral (educational or industry consortium) site.  Right now
the XML community is beginning to fragment.  Most of the XML
we see out on the net isn't valid.  And what little of it is
valid points to DTDs via in-house FPIs or relatively transient
URIs.  To outsiders, this makes XML appear unportable and un-
stable, and encourages them to focus merely on well-formedness
rather than on independently verifiable XML code.

By placing a solid, fast, basic "commodity" validator online,
and by coupling it with an FPI <->URI translation service,
we believe that we can help solidify people's notions of what
XML is and can do, and help provide the XML community with
stable tools to encourage verification and validation.

Thus far we have implemented an original, very fast XML 1.0
validation system.  It appears still to be the best publicly
available XML validator on the Web.  This validator has been
in beta testing for about two months now.  The source code for
this validator is written using stock tools (ANSI C, YACC,
Lex) and will compile in a number of different environments.
Ideally, this code should be released for free nonprofit use,
to encourage its use and improvement.

If our validator ends up being replaced by more sophisticated
commercial code, so much the better:  It will still have served
its purpose during this critical time.

STG has also designed an FPI registration system that:

  0) allows people to register FPI <-> URI associations (and
     that periodically checks the URIs for reachability, and
    sends mail to the FPI registrar if they aren't reachable)

  1) uses certificates where appropriate (e.g., if someone
     wants to register an FPI, they can't do it as, say, Sun
     Microsystems, unless they have a cert to back their claim
     up)

  2) that allows a registrar to grant maintenance privileges
     by password or IP address to others for specific FPIs (or
     to transfer "ownership" of an FPI to someone else entire-
     ly)
```

R-754

```
  3) that uses simple HTML-based web forms and portable CGI
     scripts to accomplish its work

This FPI registration system is now undergoing alpha testing
here at STG.  I expect that we could have something useful on-
line in two months.

Given the disunity of the XML community, the relatively small
amount of valid XML that there is out there, and the trouble
we are all having figuring out what to do with XML FPIs, STG
believes it would be beneficial to put up an XML validator,
coupled with an FPI registration system - and to do it from
some neutral (but well-known, authoritative) site like OASIS.

[Richard Goerwitz]
```

References:

- Scholarly Technology Group Home Page
- STG XML Validation Service
- STG XML Validation Service, Technical Report
- Xmlparse Unix Manpage
- STG SGML & HTML Document Validation Service (HTML, TEILite, CISDoc, IE, Netscape, etc.)

XSchema

- Name resolution work is under design in the context of the XSchema and XCatalog work; see the XML-DEV list.
- In July 1998, John Cowan posted a proposal to the XML-DEV mailing list for XCatalogs - "a system based on SGML/ Open catalogs (Socats) for translating public identifiers to system identifiers in XML." According to the proposal, XCatalogs are meant to be "Web resources (anything from local files on up) which contain mappings from public identifiers to system identifiers, plus references to other XCatalogs. They [would] come in two syntaxes: one which is a subset of Socat syntax, and one which is an XML document instance." The proposed XCatalog spec was revised by John Cowan to draft version 0.2 on about August 3, 1998; "the next version will hopefully be an Internet-Draft with full references." See http://www.ccil.org/~cowan/XML/XCatalog.html, and local archive copy, 980803

SCHEMA.NET

- SCHEMA.NET - "The premier site for XML DTDs and other schemata will be coming here by the end of July. If you'd like a central repository to store and make available your XML DTDs and other schemata, I am willing to host or mirror your material here on schema.net. If you are interested, email jtauber@jtauber.com. Last updated 1998-07-22."

As of November 12, 1998, the site had "Categories (9x), Other XML Schema Languages, Entity Sets, and Catalog Entries and Delegation."

R-755

Categories:

General
--begin general
DTDs for the description of general information that defies classification elsewhere on this site
:-)

ibtwsh: Itsy Bitsy Teeny Weeny Simple Hypertext DTD

"This is an XML DTD which describes a subset of HTML 4.0 for embedded use within other XML DTDs ... It is often convenient for XML documents to have a bit of documentation somewhere in them. In the absence of a DTD like this one, that documentation winds up being #PCDATA only, which is a pity, because rich text adds measurably to the readability of documents. By incorporating this DTD by reference (as an external parameter entity) into another DTD, that DTD inherits the capabilities of this one. Using HTML-compatible elements and attributes allows the documentation to be passed straight through to HTML renderers."

ibtwsh.dtd
--end general
Web, Internet, Networks
---begin web
DTDs for the description of web pages and web sites and for certain Internet networking applications.

The XML Bookmark Exchange Language (XBEL)

"The XML Bookmark Exchange Language, or XBEL, is an Internet "bookmarks" interchange format. It was designed by the Python XML Special Interest Group on the group's mailing list.

"The original intent was to create an interesting, fun project which was both useful and would demonstrate the Python XML processing software which was being developed at the time. Mark Hammond contributed the original idea, and other members of the SIG chimed in to add support for their favorite browser features. After debate which ranged far afield from the original idea, compromises were reached which allow XBEL to be a useful language for describing bookmark data for a range of browsers, including the major browsers and a number of less widely used browsers."

R-756

XBEL page on python.org

XCatalog
http://www.ccil.org/~cowan/XML/XCatalog.html
"This is a proposal for XCatalogs, a system based on SGML/Open catalogs (Socats) for translating XML public identifiers to XML system identifiers, which are Uniform Resource Identifiers."

XCatalog proposal

Extensible Log Format (XLF)

XLF Initiative Base

Process Interchange Format XML (PIF-XML)

From the PIF-XML page:
"The goal of this effort is to provide an XML version of the Process Interchange Format (PIF). The goal assumes the stated goals of PIF."

PIF-XML Home Page

WebBroker: Distributed Object Communication on the Web

From NOTE to W3C
"This [NOTE to W3C] discusses XML based mechanisms for distributed object communication on the Web."

NOTE to W3C on WebBroker

XML-RPC

"XML-RPC is a Remote Procedure Calling protocol that works over the Internet."

XML-RPC Specification

Channel Definition Format (CDF)

From the abstract of the specification:
"The Channel Definition Format is an open specification that permits a web publisher to offer frequently updated collections of information, or channels, from any web server for automatic delivery to compatible receiver programs on PCs or other information appliances."
Developed by Microsoft.

Specification (version 1.01; 1998-04-01)
Reference information for Channel Definition Format (CDF) elements used with Active Channels, Active Desktop items, and Software Update Channels.

R-757

WebDAV: Distributed Authoring and Versioning on the World Wide Web

WebDAV IETF Working Group Web Page

HTTP Duplication and Replication Protocol (DRP)

From NOTE to W3C
"The goal of the DRP protocol is to significantly improve the efficiency and reliability of data distribution over HTTP. ...[It] was designed to efficiently replicate a hierarchical set of files to a large number of clients."

NOTE to W3C on DRP

Wireless Markup Language (WML)

Part of the Wireless Application Protocol (WAP) work, the Wireless Markup Language "is intended for use in specifying content and user interface for narrowband devices, including cellular phones and pagers." - From specification

Wireless Application Protocol (WAP) Forum
WML Specification (1998-04-30) [PDF]

DMTF Common Information Model (CIM)

"The Desktop Management Task Force (DMTF) is the industry consortium chartered with development, support and maintenance of management standards for PC systems and products, including DMI and CIM."

"The DMTF has developed a Common Information Model (CIM) to take advantage of object-based management tools and provide a common way to describe and share management information enterprise-wide. Using HMMS as an input, the new model can be populated by DMI 2.0 and other management data suppliers, including SNMP and CMIP, and implemented in multiple object-based execution models such as JMAPI, CORBA and HMM. CIM will enable applications from different developers on different platforms to describe and share management data, so users have interoperable management tools that span applications, systems and networks, including the Internet."

DMTF CIM XML Home Page

R-758

--end web
Software
---software
DTDs for the description of software and related information.

Open Software Description (OSD)
http://www.microsoft.com/standards/osd/default.asp
From the press release:
"The Open Software Description (OSD) specification provides a data format or vocabulary to
describe software components, their versions, their underlying structure and their relationships to
other components."

OSD Specification (1997-08-11)
Microsoft Press Release

UML eXchange Format
http://www.yy.cs.keio.ac.jp/~suzuki/project/uxf/
From web site
"This project addresses how UML (Unified Modeling Language) models can be interchanged and
proposes an application-neutral format called UXF (UML eXchange Format), which is an exchange
format for UML models based on XML (Extensible Markup Language). It is a format powerful enough
to express, publish, access and exchange UML models, and a natural extension from the existing
Internet environment. It serves as a communication vehicle for developers, and as a well-structured
data format for development tools. With UXF, UML models can be distributed universally."

UXF Web Site

UML-Xchange
http://www.cam.org/~nrivard/uml/umlxchng.html
From web site
"UML-Xchange is a SGML DTD for exchanging data models between CASE tools that use the UML
language. All of the six kinds of UML diagrams are supported."

UML-Xchange Web Site

The CDIF XML-based Transfer Format
http://www.cdif.org/overview/XmlSyntax.html
CDIF is "a group of tool vendors, users, and system integrators sharing the vision that modelling
tools, such as CASE tools, should understand each other and interoperate seamlessly."

They are working on an XML-based transfer format.

R-759

CDIF-XML Page
--end software

Metadata, Archival, Genealogy
---meta, archiv, geneal
DTDs for metadata (data about data eg authorship, keywords, relationship to other data), the description of archival information and genealogical information.

Resource Description Framework (RDF)

From the W3C RDF Page:
"The Resource Description Framework (RDF) is a specification currently under development within the W3C Metadata activity. RDF is designed to provide an infrastructure to support metadata across many web-based activities. RDF is the result of a number of metadata communities bringing together their needs to provide a robust and flexible architecture for supporting metadata on the Internet and WWW. Example applications include sitemaps, content ratings, stream channel definitions, search engine data collection (web crawling), digital library collections, and distributed authoring.

RDF will allow different application communities to define the metadata property set that best serves the needs of each community. RDF will provide a uniform and interoperable means to exchange the metadata between programs and across the Web. Furthermore, RDF will provide a means for publishing both a human-readable and a machine-understandable definition of the property set itself.

RDF will use XML as the transfer syntax in order to leverage other tools and code bases being built around XML."

RDF at W3C (official)
RDF Model and Syntax Working Draft (1998-07-20)
RDF Schemas Working Draft (1998-04-09)
W3C Metadata activity
Frequently asked questions
RDF Made (Fairly) Easy
RDF-DEV mailing list
A Discussion of the Relationship Between RDF-Schema and UML (W3C NOTE)
RDF at DSTC
Introduction to RDF Metadata
Metadata Architecture

R-760

W3C Data Formats
RDF Implementation in Java from IBM's Alphaworks

Meta Content Framework (MCF)

From the abstract of the specification:
The MCF specification "provides the specification for a data model for describing information organization structures (metadata) for collections of networked information. It also provides a syntax for the representation of instances of this data model using XML, the Extensible Markup Language."
Developed by Netscape and others.

Meta Content Framework Using XML (1997-06-06)
http://www.w3.org/TR/NOTE-MCF-XML.html
An MCF Tutorial

Web Interface Definition Language (WIDL)
http://www.webmethods.com/technology/widl_description.html
"... a meta-data syntax implemented in XML that defines Application Programming Interfaces (APIs) to web data and services."
Developed by webMethods for their automation technology.

Description of WIDL
WIDL: Application Integration with XML

IMS Metadata Specification
http://www.imsproject.org/md_overview.html
From overview
The IMS Meta-data Specification is derived from extensive collaborations, requirements meetings, focus groups and research related to the development of meta-data specifications to support online learning. Groups included in the requirements process included teachers, instructional designers, cognitive psychologists, digital library experts, administrators of educational institutions, software developers, content developers, and meta-data experts.

The basic goals of the specification are to support the following:

- Effective discovery via the Internet of high quality materials for a particular educational or training purpose.
- Management of materials, including intellectual property rights, commerce, and customization of learning experiences.

Overview

Encoded Archival Description (EAD)

R-761

A Library of Congress standard for encoding archival finding aids.

EAD Page at Library of Congress

GedML: Genealogical Data in XML

A DTD for encoding genealogical data sets in XML. Based on GEDCOM which is a wide-spread data format for genealogical data interchange.
http://home.iclweb.com/icl2/mhkay/gedml.html
GedML Web Page
--end meta, archiv

Multimedia, Graphics, Speech
--mult
DTDs for graphics, speech, audio, video, etc and the integration of these.

Vector Markup Language (VML)
http://www.w3.org/TR/1998/NOTE-VML
"VML is an application of Extensible Markup Language (XML) 1.0 which defines a format for the encoding of vector information together with additional markup to describe how that information may be displayed and edited."

NOTE to W3C on VML

Synchronized Multimedia Integration Language (SMIL)
http://www.w3.org/AudioVideo/Activity.html
"SMIL (Synchronized Multimedia Integrated Language) is an open World Wide Web Consortium (W3C) Recommendation for the stylistic layout of multimedia presentations. SMIL defines the mechanism that authors can use to compose a multimedia presentation, combining audio, video, text, graphics and then precisely synchronize where on the screen and when these media are presented to the viewer."

W3C Audio, Video and Synchronized Multimedia Overview
Synchronized Multimedia Activity at W3C
Press Release: The World Wide Web Consortium Issues SMIL 1.0 as a W3C Recommendation
Synchronized Multimedia Integration Language (SMIL) 1.0 Specification
SMIL Information at RealNetworks
Displaying SMIL Basic Layout with a CSS2 Rendering Engine (W3C NOTE)
Internet Draft on application/smil Media Type

Precision Graphics Markup Language (PGML)
http://www.w3.org/TR/1998/NOTE-PGML
"The Precision Graphics Markup Language (PGML) is a 2D scalable graphics language designed

R-762

to meet both the simple vector graphics needs of casual users and the precision needs of graphics artists. PGML uses the imaging model common to the PostScript language and Portable Document Format (PDF); it also contains additional features to satisfy the needs of Web applications."

NOTE to W3C (1998-04-10)

Java Speech Markup Language (JSML)

The Java Speech Markup Language (JSML) is used by applications to annotate text input to Java Speech API speech synthesizers. The JSML elements provide a speech synthesizer with detailed information on how to say the text. JSML includes elements that describe the structure of a document, provide pronunciations of words and phrases, and place markers in the text. JSML also provides prosodic elements that control phrasing, emphasis, pitch, speaking rate, and other important characteristics. Appropriate markup of text improves the quality and naturalness of the synthesized voice. JSML uses the Unicode character set, so JSML can be used to mark up text in most languages of the world.

JSML Specification
http://java.sun.com/products/java-media/speech/forDevelopers/JSML/
VoxML

"The VoxML markup language for voice applications allows developers to simply and easily add speech interfaces to their Web applications or content."
http://voxml.mot.com/
Motorola's VoxML Site
end mult
Commerce, Finance, Business Information
-comm, fin. buss
DTDs for financial transactions and the interchange of business information.

Open Financial Exchange

"Open Financial Exchange is a data format designed to represent financial information exchanged between an online financial services server and a client software product. This financial data is sent back and forth between the client and server via the Internet using the Open Financial Exchange format.

The Open Financial Exchange specification, enables financial institutions and

R-763

brokerage firms to
implement online connectivity for both personal financial management (PFM) software products like
Microsoft Money or Intuit's Quicken and to build dynamic web sites. Open Financial Exchange
supports transactions for banking, credit, brokerage and mutual fund markets. It is designed to
support a wide range of financial activities including consumer and small business banking;
consumer and small business bill payment; investments, including stocks, bonds, mutual funds."

OFX: Home Page
http://www.ofx.net/
XML/EDI
http://www.xmledi.net/
From the XML/EDI Group Home Page:
"XML/EDI provides a standard framework/format to describe different types of data -- for example, an
invoice, healthcare claim, project status -- so that the information be it in a transaction, catalog or a
document in a workflow can be searched, decoded, manipulated, and displayed consistently and
correctly by implementing EDI dictionaries. Thus by combining XML and EDI we create a new
powerful paradigm!"

XML/EDI Group

Open Trading Protocol (OTP)

OTP Website
http://www.otp.org:8080/
Information & Content Exchange (ICE)
http://www.vignette.com/Products/ice/Item/0,1669,5226,00.html (FAQ)
FAQ
--end com, fin, bus

Scientific, Technical
--- begin scientif
DTDs for scientific and technical information.

W3C Math Overview
Press Release: W3C Issues MathML as a Recommendation
MathML Recommendation

Chemical Markup Language (CML)

From the CML Home Page:
"Chemical Markup Language is a radical new venture in molecular information and provides a
simple yet powerful way to manage a very wide range of problems with a single

R-764

language."
Developed by Peter Murray-Rust.

CML Site
http://www.xml-cml.org/
Bioinformatic Sequence Markup Language (BSML)
http://www.topogen.com/sbir/rfc.html
Request for Comment

Telecommunication Interchange Markup (TIM)
http://www.atis.org/atis/tcif/ipi/5tc60hom.htm
Telecommunications Industry Forum - Information Products Interchange (TCIF - IPI) Committee Home Page
----end scientif

Education
---edication
DTDs for the description of educational material.

IMS Metadata Specification
http://www.imsproject.org/md_overview.html
From overview
The IMS Meta-data Specification is derived from extensive collaborations, requirements meetings, focus groups and research related to the development of meta-data specifications to support online learning. Groups included in the requirements process included teachers, instructional designers, cognitive psychologists, digital library experts, administrators of educational institutions, software developers, content developers, and meta-data experts.

The basic goals of the specification are to support the following:

Effective discovery via the Internet of high quality materials for a particular educational or training purpose.
Management of materials, including intellectual property rights, commerce, and customization of learning experiences.

Overview

Tutorial Markup Language (TML)
http://www.ilrt.bris.ac.uk/mru/netquest/tml/about/aboutlang.html
TML Language Specification
---end Education

Language, Knowledge Representation
---lang and KR:
"DTDs for knowledge representation and the description of linguistic information."
4x on Thursday

Translation Memory eXchange (TMX)

R-765

TMX Specification
http://www.lisa.org/tmx/tmx.htm
Ontology Markup Language (OML)
http://asimov.eecs.wsu.edu/WAVE/Ontologies/OML/OML-DTD.html
OML Web Page
http://asimov.eecs.wsu.edu/WAVE/Ontologies/CKML/CKML-DTD.html
Conceptual Knowledge Markup Language (CKML)

CKML Web Page

OpenTag
http://www.opentag.org/otspecs.htm
OpenTag Specification

----
http://www.schema.net/otherschemata/">Other XML Schema Languages

Alternative XML schema languages to XML DTDs.

XML-Data

From the abstract of the specification:
XML-Data is "a specification ... for exchanging structured and networked data on the Web. This specification uses XML, the Extensible Markup Language, for describing data, as well as data about data. We expect XML-Data to be useful for a wide range of applications, such as describing database transfers, digital signatures, or remotely-located Web resources."
Developed by Microsoft and others.
Now superceded by DCD (see below)

Specification for XML-Data (1997-12-11)

XSchema

From the XSchema page
"The XSchema specification, when complete, will provide a means for XML developers to describe their XML document structures using XML document syntax."

Simon St.Laurent's XSchema page

Document Content Description for XML

From the NOTE to W3C
"This document proposes a structural schema facility, Document Content Description (DCD), for specifying rules covering the structure and content of XML documents. The DCD

R-766

proposal
 incorporates a subset of the XML-Data Submission [XML-Data] and expresses it in a way which is
 consistent with the ongoing W3C RDF (Resource Description Framework) [RDF] effort; in particular,
 DCD is an RDF vocabulary. DCD is intended to define document constraints in an XML syntax;
 these constraints may be used in the same fashion as traditional XML DTDs. DCD also provides
 additional properties, such as basic datatypes."

NOTE to W3C on dcd

Schema for Object-oriented XML (SOX)

From the NOTE to W3C
"SOX provides an alternative to XML DTDs for modeling markup relationships to enable more
 efficient software development processes for distributed applications. SOX also provides basic
 intrinsic datatypes, an extensible datatyping mechanism, content model and attribute interface
 inheritance, a powerful namespace mechanism, and embedded documentation. As compared to
 XML DTDs, SOX dramatically decreases the complexity of supporting interoperation among
 heterogenous applications by facilitating software mapping of XML data structures, expressing
 domain abstractions and common relationships directly and explicitly, enabling reuse at the
 document design and the application programming levels, and supporting the generation of
 common application components."

NOTE to W3C on SOX

http://www.schema.net/entities/">Entity Sets

The following are character entity sets in common use:

ISO Entities

Courtesy of Rick Jelliffe

ISO 8879:1986//ENTITIES Added Latin 1//EN//XML
ISO 8879:1986//ENTITIES Added Latin 2//EN//XML
ISO 8879:1986//ENTITIES Numeric and Special Graphic//EN//XML
ISO 8879:1986//ENTITIES Publishing//EN//XML
ISO 8879:1986//ENTITIES General Technical//EN//XML
ISO 8879:1986//ENTITIES Diacritical Marks//EN//XML
ISO 9573-15:1993//ENTITIES Greek Letters//EN//XML
ISO 9573-15:1993//ENTITIES Monotoniko Greek//EN//XML

R-767

ISO 8879:1986//ENTITIES Greek Symbols//EN//XML
ISO 8879:1986//ENTITIES Alternative Greek Symbols//EN//XML

http://www.schema.net/catalog/">Catalog Entries and Delegation

I have set up an SGML Open Catalog at http://www.schema.net/public-text/catalog.soc so that people can resolve public identifiers without having to maintain their own catalogs. By including a delegation to this catalog or pointing to it directly, processors that support SGML Open Catalogs can make use of the entries included.

As well as including direct entries, the catalog can act as a root for further delegation. If you produce public text (eg DTDs) and have set up your own online catalog, I am more that willing to add a DELEGATE entry in my catalog so that processors using my catalog will automatically use yours when necessary. Just contact me with your owner identifier and the URL of your catalog.

NOTE: This is an experimental service. I don't guarantee anything, but please feel free to send me your feedback.

Tauber's Current catalog file

```
-- SGML Open Catalog at SCHEMA.NET        --

-- If you would like local public text or --
-- a delegation added, please email       --
-- jtauber@jtauber.com                    --

-- Local public text --
PUBLIC "ISO 8879:1986//ENTITIES Added Latin 1//EN//XML" "http://www.schema.net/
public-text/ISOlat1.pen"
PUBLIC "ISO 8879:1986//ENTITIES Added Latin 2//EN//XML" "http://www.schema.net/
public-text/ISOlat2.pen"
PUBLIC "ISO 8879:1986//ENTITIES Numeric and Special Graphic//EN//XML" "http://www.
schema.net/public-text/ISOnum.pen"
PUBLIC "ISO 8879:1986//ENTITIES Publishing//EN//XML" "http://www.schema.net/public-
text/ISOpub.pen"
PUBLIC "ISO 8879:1986//ENTITIES General Technical//EN//XML" "http://www.schema.net/
public-text/ISOtech.pen"
PUBLIC "ISO 8879:1986//ENTITIES Diacritical Marks//EN//XML" "http://www.schema.net/
public-text/ISOdia.pen"
PUBLIC "ISO 9573-15:1993//ENTITIES Greek Letters//EN//XML" "http://www.schema.net/
```

R-768

```
public-text/ISOgrk1.pen"
PUBLIC "ISO 9573-15:1993//ENTITIES Monotoniko Greek//EN//XML" "http://www.schema.net/
public-text/ISOgrk2.pen"
PUBLIC "ISO 8879:1986//ENTITIES Greek Symbols//EN//XML" "http://www.schema.net/
public-text/ISOgrk3.pen"
PUBLIC "ISO 8879:1986//ENTITIES Alternative Greek Symbols//EN//XML" "http://www.
schema.net/public-text/ISOgrk4.pen"

-- Delegation --
DELEGATE "-//W3C" "http://validator.w3.org/sgml-lib/catalog"
DELEGATE "+//IDN python.org" "http://www.python.org/topics/xml/dtds/catalog"
```

Graphic Communications Association Registry of Owner Identifiers

- GCA registry of owner identifiers - "We have established this page in order to provide a central repository of owner identifiers for use in formal public identifiers and links to SGML public text. Our hope is to facilitate the exchange of information, provide better support for the industry and further promote the use of the International Standard ISO 8879: Standard Generalized Markup Language (SGML) by encouraging the re-use of already created public text."
- Registration process for public text owner identifiers
- List of registered owners of public text
- Application form

XML.com

- XML.com and O'Reilly were [at one point] reported to be designing an XML DTD registry. Status?

XMLrepository.com

"This XML repository is the staging/collection area for all XML related technology. Tag sets, document type definitions (DTDs), and Extensible Stylesheet Language (XSL) templates, and other schema needed for effective communication will be collected here and eventually moved to their own sites, i.e. XSL stylesheets to XSLstylesheets.com, DTDs to DTDstylesheets.com, etc. A special site, DTDML.com, will be the official site for storing mathML, chemML, musicML, electronicML, scienceML, etc."

Contact: admin@xmlrepository.com

- Home Page - http://xmlrepository.com/
- http://xmlshareware.com/
- XML Related Domain Names Are For Sale.

Software

- "Resolving Formal Public Identifiers." - A pilot service to resolve a Formal Public Identifier (FPI). From Peter Flynn, University College Cork. [local archive copy, display form only]
- SAX EntityResolver (basic interface for resolving entities). - See the documentation, and the note by David Megginson.

World Wide WEAVE.com

- World Wide WEAVE.com - "Weave allows people to publicize new XML-based web sites and documents. Weave contains only sites that provide content in XML-based markup." Taxonomy [1999-03-01]: Business & Commerce, Computing & Technology, Education, Entertainment, Finance, Health, Hobbies, Fine Arts & Literature, News & Media, People, Real Estate, Reference, Regional, Sports & Leisure, Travel, XML Development.

Some Reference Documents

- Namespaces in XML W3C Recommendation, REC-xml-names-19990114.
- "URI Resolution Services Necessary for URN Resolution." By M. Mealling and R. Daniel, Jr. Network Working Group Request for Comments: 2483. January 1999. [local archive copy]
- The Best of Distributed Objects By Nelson Minar, Mark Baker, Joe Kiniry, and Ron Resnick.
- Persistent Identifiers on the Digital Terrain By Sandra Payette
- Unique Identifiers: a brief introduction By Brian Green and Mark Bide
- Unique Identifiers for Digital Information - check this
- Universal Document Naming Systems: Links check
- Identifiers and Their Role in Networked Information Applications by Clifford Lynch
- Universal Resouree Names
- Universal Resource Characteristics (URCs)
- Uniform Resource Names - A Progress Report
- The Path URN Specification
- Namespace and Resource Identifiers: Specs and White Papers
- Resolution of Uniform Resource Identifiers using the Domain Name System. Ron Daniel. draft-ietf-urn-naptr-05.txt, 22 May, 1997
- "ISO Formal Public Identifiers as a URN Namespace." 12/4/1996. By Ron Daniel.
- URN Namespaces - LANL
- Naming and Addressing: URIs (W3C)
- Uniform Resource Identifiers (URI): Generic Syntax Request for Comments: 2396. August 1998.
- IETF Uniform Resource Identifiers (URI) Working Group
- Uniform Resource Names (urn) - IETF Charter
- Uniform Resource Identifiers (URI): GenericSyntax (RFC 2396) - Roy Fielding
- "Representing non-ASCII Characters in URIs and Extended URIs"
- "Handling Internationalized Query Components in URLs"
- "Registration Procedures for URL Scheme Names"
- "Some Alternatives to draft-ietf-urlreg-proeedures-03"
- "Guidelines for new URL Sehemes"

R-770

- Internet Domain Name System Management
- Universal Unique Identifier

Syntax Issues

URIs (URNs) allow characters not directly allowed in XML, and the XML PI allows characters not generally allowed in the SGML FPI. Etc.

```
SGML FPI has eleven "special" chars allowed in md, XML PI allows nineteen
an additional 8x, =  ; ! * # @ $ _ %

PI in XML:

[12] PubidLiteral ::=
        '"' PubidChar* '"' | "'" (PubidChar - "'")* "'"
[13] PubidChar ::=
        #x20 | #xD | #xA | [a-zA-Z0-9] | [-'()+,./:=?;!*#@$_%]

- - - - - -

FPI in 8879:

[77] minimum data =
      minimum data character [78] *

[78] minimum data character =
      RS | (13) CR
      RE | (10) LF
      SPACE | (32) space
      LC Letter | a-z
      UC Letter | A-Z
      Digit | 0-9
      Special '( )+,-./:=?

[79] formal public identifier =
      owner identifier [80] ,
      "//" ,
      text identifier [84]

[80] owner identifier =
      ISO owner identifier [81] |
      registered owner identifier [82] |
      unregistered owner identifier [83]

[81] ISO owner identifier =
      minimum data [77]

[82] registered owner identifier =
      "+//" ,
```

R-771

```
      minimum data [77]

[83] unregistered owner identifier =
      "-//" ,
      minimum data [77]

[84] text identifier =
      public text class [86] ,
      SPACE , (32) space
      unavailable text indicator [85] ?
      public text description [87] ,
      "//" ,
      ( public text language [88] |
      public text designating sequence [89] ),
      ( "//" ,
      public text display version [90] )?

[87] public text description =
      ISO text description [87.1] |
      minimum data [77]

[87.1] ISO text description =
      minimum data [77]

[88] public text language =
      name [55]

[90] public text display version =
      minimum data [77]
```

Notes:

http://www.lists.ic.ac.uk/hypermail/xml-dev/9810/0308.html
Re: A call for open source DTDs

Elliotte Rusty Harold (elharo@sunsite.unc.edu)
Thu, 15 Oct 1998 11:17:57 -0400

Messages sorted by: [date][thread][subject][author]
Next message: Elliotte Rusty Harold: "RE: A call for open source DTDs"
Previous message: Scott Vanderbilt: "Re: A call for open source DTDs"
Next in thread: Toby Speight: "Re: A call for open source DTDs"
Reply: Toby Speight: "Re: A call for open source DTDs"

Let me elaborate a little on the problem. Let us suppose we have a DTD for plumbing. This DTD is copyright 1998 International Plumbers Association. Can I legally place the DTD is an XML document of my own creation? The answer is no. That would be the same as including an entire poem or other work in my document rather than quoting a part of it.

R-772

Can I place the DTD in a separate file on my web server and reference it like this:

<!DOCTYPE document SYSTEM "http://sunsite.unc.edu/xml/plumbing.dtd">

Again, legally, the answer is no. I cannot legally place the copyrighted document on my server any more than I can copy a copyrighted HTML file from another web site onto my own.

I can, however, do this:

<!DOCTYPE document SYSTEM "http://www.iap.org/xml/plumbing.dtd">

This relies on the International Association of Plumburs not changing the URL of the plumbing DTD, not changing the DTD itself, and maintaining a web server that's fast and accessible independently of the state of my web server. And it completely fails for offline documents. So this isn't a good solution.

Is open source a solution? Maybe, especially if the DTD is external to the document. However, standard open source licenses like the GPL are problematic because they would seem to imply that if the DTD is included with the document itself, then the entire document must be open source. They are one file, after all. Not even Richard Stallman tries to make all programs compiled with gcc, open source. Neither should using an open source DTD taint the document the DTD validates. So if we want open source we need a new kind of open source license that's clearer about the distinction between DTDs and documents, even though they may be present in the same file.

The simplest solution is to simply declare that the DTD is in the public domain. This works well with existing systems and allows anyone to use the DTD any way they need to. The only potential downside I see to this is that there may be some standardization problems if people are allowed to change the DTD willy-nilly. Long-term I suspect we'll develop some standard licensing language that allows unlimited reuse, but only if the name is changed, perhaps something like Perl's artistic license where you can do anything you want with it as long as you don't call it Perl.

In any case, the main thing I want to bring up is to make sure DTD authors think about these things when writing copyright statements. If people are going to use your DTD, they absolutely must be able to republish it. Without special permission, standard copyright prevents that.

```
+-----------------------+------------------------+--------------------+
| Elliotte Rusty Harold | elharo@sunsite.unc.edu | Writer/Programmer |
+-----------------------+------------------------+--------------------+
| XML: Extensible Markup Language (IDG Books 1998) |
| http://www.amazon.com/exec/obidos/ISBN=0764531999/cafeaulaitA/ |
+----------------------------------+-----------------------------------+
| Read Cafe au Lait for Java News: http://sunsite.unc.edu/javafaq/ |
```

R-773

```
| Read Cafe con Leche for XML News: http://sunsite.unc.edu/xml/ |
+----------------------------------+----------------------------------+

xml-dev: A list for W3C XML Developers. To post, mailto:xml-dev@ic.ac.uk
Archived as: http://www.lists.ic.ac.uk/hypermail/xml-dev/
To (un)subscribe, mailto:majordomo@ic.ac.uk the following message;
(un)subscribe xml-dev
To subscribe to the digests, mailto:majordomo@ic.ac.uk the following message;
subscribe xml-dev-digest
List coordinator, Henry Rzepa (mailto:rzepa@ic.ac.uk)
```

R-774

Exhibit Q. Microsoft Corp., XML: Enabling Next-Generation Web Applications (Apr. 3, 1998) accessed at http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/dnarxml/html/xmlwp2.asp on July 21, 2007, 15 pp.

XML: Enabling Next-Generation Web Applications

Microsoft Corporation

April 3, 1998

Contents

Abstract

Extensible Markup Language (XML) provides a significant advance in how data is described and exchanged by Web-based applications using a simple, flexible standards-based format. Hypertext markup language (HTML) enables universal methods for *viewing* data; XML provides universal methods for *working* directly with data.

Introduction

Internet protocol (IP), the Hypertext Markup Language (HTML) and the Hypertext Transport Protocol (HTTP) have revolutionized the manner in which information is distributed, displayed and searched for. Organizations rapidly embraced browsers and search engines with the creation of corporate intranets, then extended these to customers, suppliers, and business partners with extranets.

Extensible Markup Language (XML), which complements HTML, promises to increase the benefits that can be derived from the wealth of information found today on IP networks around the world. This is because XML provides a uniform method for describing and exchanging structured data. The ability to describe structured data in an open text-based format and deliver this data using standard HTTP protocol is significant for two reasons. XML will facilitate more precise declarations of content and more meaningful search results across multiple

R-776

platforms. And once the data is located it will enable a new generation of viewing and manipulating the data.

HTML can be used to tag a word to be displayed in bold or italic; XML provides a framework for tagging structured data. An XML element can declare its associated data to be a retail price, a sales tax, a book title, the amount of precipitation, or any other desired data element. As XML tags are adopted throughout an organization's intranet, and by others across the Internet, there will be a corresponding ability to search for and manipulate data regardless of the applications within which it is found. Once data has been located, it can be delivered over the wire and presented in a browser such as Microsoft® Internet Explorer 4.0 in any number of ways, or it can be handed off to other applications for further processing and viewing.

XML is a subset of Standard Generalized Markup Language (SGML) that is optimized for delivery over the Web; it is defined by the World Wide Web Consortium (W3C), ensuring that structured data will be uniform and independent of applications or vendors. This resulting interoperability kick-starts a new generation of business and electronic-commerce Web applications.

Microsoft Corporation is working with the W3C and other organizations to create a powerful standard definition of XML to significantly advance how people can take advantage of Web-based data. HTML enables a universal method for *displaying* data; XML provides a universal method for *describing* data. There is a rapidly growing need to go beyond simply *viewing* data, as is the case with Web browsers today. People need to be able to work with and manipulate the data. And this is what XML allows.

Microsoft has implemented support for the XML standard within Internet Explorer 4.0 and will use XML in its next release of Microsoft Office and other products. This commitment to XML is important because corporations are increasingly moving from classic client/server two-tier application models to three-tier models, in which a browser front end interacts with a middle-tier Web server, which in turn communicates with a back-end database server for storage. This three-tier architecture has several benefits over client/server models, including easier scalability and better security. And XML will make possible richer implementation of such models through structured data exchanged over HTTP.

The power and beauty of XML is that it maintains the separation of the user interface from structured data, allowing the seamless integration of data from diverse sources. Customer information, purchase orders, research results, bill payments, medical records, catalog data and other information can be converted to XML on the middle tier, allowing data to be exchanged online as easily as HTML pages display data today. Data encoded in XML can then be delivered

R-777

over the Web to the desktop. No retrofitting is necessary for legacy information stored in mainframe databases or documents, and because HTTP is used to deliver XML over the wire, no changes are required for this function.

Once the data is on the client desktop, it can be manipulated, edited, and presented in multiple views, without return trips to the server. Servers now become more scalable, due to lower computational and bandwidth loads. Also, since data is exchanged in the XML format, it can be easily merged from different sources.

XML is valuable to the Internet as well as large corporate intranet environments because it provides interoperability using a flexible, open, standards-based format, with new ways of accessing legacy databases and delivering data to Web clients. Applications can be built more quickly, are easier to maintain, and can easily provide multiple views on the structured data.

What is XML?

Extensible Markup Language is a text-based format that lets developers describe, deliver and exchange structured data between a range of applications to clients for local display and manipulation. XML also facilitates the transfer of structured data between servers themselves. Vast stores of legacy information exist today, distributed across disparate, incompatible databases. XML allows the identification, exchange and processing of this data in a manner that is mutually understood, using custom formats for particular applications if needed.

XML resembles and complements HTML. XML describes data, such as city name, temperature and barometric pressure, and HTML defines tags that describe how the data should be displayed, such as with a bulleted list or a table. XML, however, allows developers to define an unlimited set of tags, bringing great flexibility to authors, who can decide which data to use and determine its appropriate standard or custom tags.

In this case, XML is used to describe a weather report:

```
<weather-report>

   <date>March 25, 1998</date>

   <time>08:00</time>

   <area>

      <city>Seattle</city>

      <state>WA</state>

      <region>West Coast</region>
```

R-778

```
    <country>USA</country>
  </area>
  <measurements>
    <skies>partly cloudy</skies>
    <temperature>46</temperature>
    <wind>
      <direction>SW</direction>
      <windspeed>6</windspeed>
    </wind>
    <h-index>51</h-index>
    <humidity>87</humidity>
    <visibility>10</visibility>
    <uv-index>1</uv-index>
  </measurements>
</weather-report>
```

This data could be displayed in many different ways, or it could be handed off to other applications for further processing. In the future, style sheets could help by transforming structured data into different HTML views for display in a browser, or even to other display formats on other platforms running other applications.

Today with XML, Document Type Definitions (DTDs) may accompany a document, essentially defining the rules of the document, such as which elements are present and the structural relationship between the elements. DTDs help to validate the data when the receiving application does not have a built-in description of the incoming data. With XML, however, DTDs are optional.

Data sent along with a DTD is known as "valid" XML. In this case, an XML parser could check incoming data against the rules defined in the DTD to make sure data was structured correctly. Data sent without a DTD is known as "well-formed" XML. Here an XML-based document instance, such as the hierarchically structured weather data above, can be used to implicitly describe itself.

R-779

With both valid and well-formed XML, XML encoded data is self-describing. The open and flexible format used by XML allows it to be employed anywhere a need exists for the exchange and transfer of information. This makes it *extremely* powerful.

For instance, XML can be used to describe information about HTML pages, or it can be used to describe data contained in business rules or objects in an electronic-commerce transaction, such as invoices, purchase orders and order forms. Since XML is separate from HTML, XML could also be added inside HTML documents. In fact, Microsoft is working with W3C to define a format by which XML-based data, or "XML islands," can be encapsulated in HTML pages. By embedding XML data inside an HTML page, multiple views could be generated from the delivered data, using the semantic information contained in the XML. Moreover, XML can be used for such compelling applications as distributed printing, database searches, and others.

Benefiting from XML

Extensible Markup Language brings so much power and flexibility to Web-based applications that it provides a number of compelling benefits to developers and users:

- More meaningful searches
- Development of flexible Web applications
- Data integration from disparate sources
- Data from multiple applications
- Local computation and manipulation of data
- Multiple views on the data
- Granular updates
- Open standards
- Format for Web delivery
- Enhances scalability
- Facilitates compression
- Support by Microsoft products
- Provides great opportunities

More meaningful searches

Data can be uniquely tagged with XML, allowing a customer to specify books *by*, rather than

R-780

about, Winston Churchill, for example. By contrast, searches using present methods would probably yield both types of books mixed together. Without XML, it is necessary for the searching application to understand the schema of each database, which describes how it is built. This is virtually impossible because every database describes its data differently. With XML, however, books could be easily categorized in a standard way by author, title, ISBN number, or other criteria. Agents could then search these identified bookstore sites in a consistent way for books *about* Winston Churchill, for example.

Development of flexible Web applications

Once data has been found, XML can be delivered to other applications, objects, or middle-tier servers for further processing. Or it can be delivered to the desktop for viewing in a browser. XML, together with HTML for display, scripting for logic, and a common object model for interacting with the data and display, provides the technologies needed for flexible three-tier Web application development.

Data integration from disparate sources

The ability to search multiple, incompatible databases is virtually impossible today. XML enables structured data from different sources to be easily combined. Software agents can be used to integrate data on a middle-tier server from back-end databases and other applications. This data can then be delivered to clients or other servers for further aggregation, processing and distribution.

Data from multiple applications

The extensibility and flexibility of XML allow it to describe data contained in a wide variety of heterogeneous applications, from describing collections of Web pages to data records. Again, since XML-based data is self-describing, data can be exchanged and processed without having a built-in description of the incoming data.

Local computation and manipulation

After being delivered to the client, data in XML format may be parsed and locally edited and manipulated, with computations performed by client applications. Users may manipulate data in various ways, rather than merely presenting it. The XML Object Model also allows data to be manipulated with scripting or other programming languages. Data computations can be performed without additional return trips to the server. Returning to the example of weather data, XML can be used to locate areas of high and low temperature and compute the average temperature, either in a given place over a period of time, or in several places at once. If barometric pressure or other data were available for every county in the country, XML could allow a national weather map to chart low and high pressure and other factors. Separating the user interface that views data from the data itself allows powerful applications, formerly found only on high-end databases, to be created naturally for the Web using a simple, flexible, open format.

R-781

Multiple views of data

Once data has been delivered to the desktop, it can be viewed in different ways. By describing structured data in a simple, open, robust, and extensible manner, XML complements HTML, which is widely used to describe user interfaces. Again, while HTML describes the appearance of data, XML describes data itself. In the case of a weather map, an average user might want to see just the high and low temperatures and precipitation data. But pilots and others might also want to know about air pressure, dew points and other information. Since display is now separate from data, having this data defined in XML allows different views to be specified, resulting in data being presented appropriately. Local data may be presented dynamically in a manner determined by client configuration, user preference or other criteria.

Granular updates

Data may be granularly updated with XML, eliminating the need to resend an entire structured data set each time a portion of the data changes. Only the changed element must be sent from the server to the client, and the changed data can be displayed without refreshing the entire user interface. Using the example of weather data, if the low temperature dropped from 35 to 34 degrees, XML would not require the entire weather map to be rebuilt, because the desired piece of information can be changed separately. Presently, an entire page must be rebuilt if even one item of data changes, even when the view remains constant. This severely limits server scalability.

Also, XML allows other data to be added, such as predicted high and low temperatures, expected precipitation, and the probability (in percent). This additional information can stream into the user's existing view, without the browser having to send a new view. If additional information such as barometric pressure is requested, it can be sent without rebuilding.

Delivery of data on the Web

Because XML is an open text-based format, it can be delivered using HTTP in the same way that HTML can today without any changes to existing networks.

Scalability

Because XML completely separates the notion of markup from its intended display, authors can embed in structured data procedural descriptions of how to produce different data views. This is an incredibly powerful mechanism for offloading as much user interaction as possible to the client computer, while reducing server traffic and browser response times. In addition, XML lets individual pieces of data be altered with only an update notice, greatly enhancing server scalability as a result of a far lower workload.

Compression

XML compresses extremely well due to the repetitive nature of the tags used to describe data structure. The need to compress XML data will be application-dependent and largely a function of the amount of data being moved between server and client. Benchmarks will be provided in the future to help determine whether compression is necessary. XML

R-782

can use the compression standard in HTTP 1.1 servers and clients.

Open standards

XML is based on proven standards-based technology optimized for the Web. Microsoft is working with the W3C XML Working Group and other leading companies to help ensure interoperability and support for developers, authors, and users on multiple systems and browsers, and to evolve the XML standard.

The XML initiative consists of three related standards:

- **XML (Extensible Markup Language).** XML 1.0 is now a recommendation, the final stage in the W3C approval process. This means that the standard is stable and can be fully embraced by Web and tools developers.
- **XSL (Extensible Style Language).** Microsoft, together with ArborText Inc. and Inso Corp., has presented a specification to W3C known as *Extensible Style Language*. XSL is used to transform XML-based data into HTML or other presentation formats. Currently the XSL standard is in the working group phase at W3C and is still under development. XSL provides a superset of the Cascading Style Sheets (CSS) language functionality and allows developers to build a presentation structure that is different from the data structure. For instance, XSL can be used to transform an XML-based purchase order number into a bulleted list in one HTML view, and into a footnote in a second HTML view. CSS may still be used for simply structured XML data but cannot present information in an order different from how it was received. XSL will also be able to generate CSS along with HTML.
- **XLL (Extensible Linking Language).** XLL is an XML linking language that provides links in XML similar to those in HTML but offers more power. With XLL, for example, linking could be multidirectional, and links could exist at an object level rather than just at a page level.
- **XML OM (XML Object Model).** In addition, the XML OM tracks the W3C standard Document Object Model (DOM) to allow programmatic access to structured data through scripting, so developers can interact with and compute on XML-based data consistently.

Supported by Microsoft Products

Microsoft Internet Explorer 4.0 allows users to edit and view XML-based data. Internet Explorer 4.0 supports a number of XML features:

- **A generalized XML parser.** The parser reads XML files and generates a hierarchically structured tree, then hands off data to viewers and other applications for processing. Microsoft has two parsers, the MSXML Parser, a high-performance, nonvalidating parser written in C++ that ships with Internet Explorer 4.0, and the MSXML Parser in Java, a validating parser available for download from the Web by developers for inclusion in their applications.
- **XML Data Source Object (XML DSO).** The XML DSO assists developers in connecting to structured XML data and supplying it to an HTML page using Dynamic HTML's data-binding facility.

R-783

- **XML Object Model.** Microsoft also supports the XML Object Model to allow programmatic access to structured data using the XML parsers, giving developers the power to interact and compute data in a consistent fashion. The XML OM tracks the W3C DOM.

With Microsoft Site Server Commerce Edition, customers can exchange data between applications in XML over HTTP using the Commerce Interchange Pipeline (CIP) infrastructure.

In the future, Microsoft Office documents will enable "round-tripping," allowing customers to save an Office document in HTML and reopen it in Office. In this case, XML data will be used to annotate HTML files, preserving Office-specific functionality.

The robust XML support included with Internet Explorer 4.0 and other Microsoft products encourages developers to publish XML-enhanced pages and increases the value of XML to developers and webmasters.

Opportunities

As an industry standard for expressing structured data, XML offers many advantages to organizations, software developers, Web sites, and end users. The opportunities will expand further as more vertical market data formats are created for key markets such as advanced database searching, online banking, medical, legal, electronic commerce, and other fields. And when sites dispense data rather than just views on data, extraordinary opportunities result.

Customer services are now migrating to Web sites from call centers and physical locations and will therefore benefit from the robust functionality of XML. And, because most of these business applications involve manipulation or transfer of data and database records, such as purchase orders, invoices, customer information, appointments, maps and such, XML will revolutionize end-user possibilities on the Internet by allowing a rich array of business applications to be implemented. In addition, information already on Web sites, whether stored in documents or databases, can be marked up using XML-based, intranet-oriented vocabularies. These vocabularies also help small and medium-sized corporations that need to exchange information between customers and suppliers.

A vital untapped market is development tools that make it easy for end users to build their own collaborative Web sites, including tools for generating XML data from legacy database information and from existing user interfaces. In addition, standard schemas could be developed for describing portfolios or other data, for example, which could use the layout, graphs and other functions of Microsoft Excel or other existing spreadsheets. Declarative and visual tools for describing XML generated from legacy databases are a powerful opportunity. Custom tools for viewing XML data can be written in the Visual Basic® development system, Java, and C++.

XML will require powerful new tools for presenting rich, complex XML data within a document; this is done by mapping a user-friendly display layer on top of a complex set of hierarchical data that can change dynamically. Possible layouts to use for XML data include collapsing outlines, PivotTable® dynamic views, and a simple sheet for each portfolio.

R-784

Web sites offer stock quotes or news articles or real-time traffic data, which can be obtained by filtering from Web broadcasts or by intelligent polling of a tree of servers replicating these sites. Information overload can be avoided with XML by writing custom rules for aging of information, for example, as is done with e-mail. XML-based tools for users to construct these rules and server and client software to execute them are a huge opportunity. A Standard Object Model could enable these functions, typically written in script, to filter incoming messages, examine stored messages, create outgoing messages, access databases and such. These agents can be written to run anywhere automatically.

XML Architecture: Building Flexible Three-Tier Web Applications

The XML language is now a W3C recommendation, the final stage in the W3C development and approval process. Because of the fully stable specification, developers can start tagging and exchanging their data in the XML format. XML offers a robust solution as the underlying architecture for data in three-tier architectures, as described below.

Three-Tiered Architecture

XML can be generated from existing databases using a scalable three-tier model. With XML, structured data is maintained separately from the business rules and the display. Data integration, delivery, manipulation and display are the steps in the underlying process.

Data Integration

Agents running on the middle tier will be able to access multiple existing databases and translate existing data into XML.

Making Data Self-Describing

Document Type Definitions describe classes of documents

Today with valid XML, DTDs are used to describe the syntax of a class of XML documents. DTDs use a different syntax from that used by XML documents and are primarily designed for the exchange of long, structured documents. They lack rich data-typing capabilities necessary to describe data contained in databases.

XML-Data schemas promise improved integration with relational databases

Schemas are a future technology outlined in the XML-Data Proposal recently submitted to the W3C. A schema is a formal specification of element names that indicates which elements are allowed in an XML document, and in what combinations. Using an XML-Data schema, the author can define which element names are permitted in his document; within each element, the author can decide which subelements, attributes and relations are allowed. In this regard, they are functionally equivalent to DTDs.

XML-Data schemas, however, are written in XML itself, allowing XML to describe its own structure. XML-Data also outlines a model for extending schemas, allowing developers to augment them with such facilities as data types, inheritance and presentation rules. For this reason, they are a powerful alternative to DTDs that promise improved integration of data stored in relational databases. With the strong data-typing facilities schemas provide, developers

R-785

could specify that the data contained in a <date> element was indeed an ISO8601 formatted date, not just a number. Developers could also define their own specialized data types. With the inheritance capability of schemas, developers could base a new specialized schema on an existing, general-purpose schema. For instance, developers could define a bookstore purchase-order schema based on a general-purpose electronic-commerce purchase-order schema.

Qualifying Data Uniquely

Namespaces

XML namespaces let developers qualify element names in a recognizable manner to avoid conflicts between elements with the same name. Elements referenced in one document, such as a purchase order, may be defined in different schemas on the Web. Namespaces ensure that element names do not conflict and clarify their origins, but do not determine how to process elements. Parsers must know what elements mean and how to process them.

Tags from multiple namespaces can be mixed, which is essential with data coming from multiple sources across the Web. With namespaces, both elements could exist in the same XML-based document instance, but could refer back to two different schemas, uniquely qualifying their semantics. For instance, in a bookstore purchase order, one <title> element could contain a book title, and another <title> element could contain the author's title.

A proposal has been submitted to the W3C to make every element name subordinate to a universal resource identifier (URI). This ensures that names remain unambiguous even if chosen by multiple authors. Just as anyone can publish his or her own Web page or view those of others, the namespace facility allows users to define private dictionaries of terms or use a public namespace of common terms.

Data Delivery, Manipulation

Because XML is an open, text-based format, it can be delivered through HTTP in the same way HTML can today. Data now on the desktop can be manipulated using the XML Object Model. Agents will also support the ability to generate XML updates, which can be sent in both directions to inform clients of changes made to data on the middle tier or database server and vice versa. Consequently, the agents will be able to receive updates from the client and send them to a storage server.

Parsing XML

Today, the XML parser in Internet Explorer 4.0 can read a string of XML data, process it, generate a structured tree and expose all data elements as objects using the XML Document Object Model (XML DOM). The parser then makes the data available for further manipulation through scripting. At this point, the data could also be handed off to other applications or objects for further processing.

Manipulating and editing data using XML object model

The XML Object Model is essentially an API that defines a standard way in which developers can interact with the elements of the XML structured tree. For example, the addChild method included in the XML Object Model allows data

R-786

sets such as "barometric pressure" to be added under existing data. Similarly, interaction using the object model could be used to change the temperature on the weather page. The object model controls how users communicate with trees and exposes all tree elements as objects, which can be accessed programmatically without any return trips to the server.

Displaying XML-Based Data in HTML

As outlined earlier, XML-based data does not contain information about how data, such as city name, temperature or barometric pressure, should be displayed. To display this data, either the Web server or the Web browser will need to move XML-based data into HTML for display, or alternatively transform XML data into HTML. Data binding is a facility in Dynamic HTML in Internet Explorer 4.0 that can be used to move XML-based data elements into HTML for display. The XML DSO can assist developers in this process, essentially merging data into an HTML presentation template. XML-based data can be moved into an HTML display with or without the assistance of the XML DSO; however, in both cases scripting is required.

In the future using XSL, developers will be able to generate HTML without complex scripting. XSL is a transformation language that defines rules for mapping structured XML data to HTML, or other display formats, using an XSL Processor. XSL could also be used to generate CSS along with HTML. (See Figure 3 above.)

A group of these rules defines a style sheet. XSL lets data be transformed into a presentation structure different from the original data structure, allowing data elements to be formatted and displayed in multiple places on a page, rearranged or removed. Many style sheets can exist for one data set, describing various delivery platforms or output devices, or many data sets can reference one style sheet. Microsoft recently released a preview XSL processor for developers to experiment with and provide feedback for the further development of the technology.

XML Vocabularies

XML vocabularies are the actual elements used in particular data formats. Vocabularies and the structural relationships between elements can be formally defined in XML DTDs or XML-Data schemas as data formats.

In intranet environments, departments will be able to define XML vocabularies quickly and easily to share data between applications, as well-formed or valid XML, without industrywide approval processes. And since XML is flexible, it can change as data description and exchange requirements evolve.

In Internet environments, the model will be to define and agree upon vocabularies for sharing information. This model has worked well in the SGML industry and is starting to work well with newly defined vocabularies. Channel definition format (CDF), for example, is a format for describing collections of pages and when these pages should be downloaded.

Here are some examples of newly defined formats:

Channel Definition Format

R-787

Channel definition format is an XML-based data format used in the Microsoft Internet Explorer 4.0 browser to describe Active Channel™ content and desktop components. It is used by thousands of content developers and millions of end users to describe collections of pages and data about pages, such as channel-bar display, download behavior, Web-page usage, and page-hit logging.

Open Software Description

Open software description (OSD) is an XML-based data format fully supported in Microsoft Internet Explorer 4.01, for advertising and installing software components over the Internet. When new versions of software become available, OSD provides a publishing mechanism to notify users. OSD can provide detailed descriptions of how to install ActiveX® Controls and Java packages and class files, adding increased functionality to facilitate setup.

Open Financial Exchange

Open Financial Exchange (OFX) is a data format used by Microsoft Money 97 and Intuit Quicken personal-finance applications to communicate with financial institutions over the Web. OFX was developed with the cooperation of Checkfree; although it is currently described using SGML, OFX will soon be based on XML and therefore accessible to users of XML-enabled Web browsers.

Electronic Data Interchange

XML can make electronic data interchange (EDI) transactions accessible to a broader set of users. EDI is already a powerful tool that has been deployed by large organizations around the globe to exchange data and support transactions. But today, EDI transactions can only be conducted between sites that have been specifically set up to exchange information using compatible systems. XML will allow data to be exchanged, regardless of the computing systems or accounting applications being used. XML should create new possibilities for how data is used, and there are many initiatives under way to move EDI to XML.

Resource Definition Framework

Resource Definition Framework (RDF), unlike XML, is not a format or language but rather more like an API. Today XML comes with its own API, or Document Object Model, which is based on the corresponding DOM for HTML. This XML API leverages the physical structure of the document, such that developers can programmatically look at the document as collections of "trees" with trunks, branches, leaves, etc. This is a great way for looking at XML as a natural language document or as bulk data, but it's not particularly good for grasping the logical structure of a document or the meanings of particular XML tags in relation to other tags. For this, the W3C is developing the Resource Definition Framework. This new object model, or API, allows developers to access the logical meaning of designated content in XML documents. One example might be information of the sort used by libraries to catalog books, such as "author," "copyright" or "related subjects."

Users can expect to see a limited form of RDF on the Web expressed using a specialized XML syntax known as the RDF Namespace. In its more general form, RDF will work with today's unadorned XML, with its native namespace, not the

R-788

constrained RDF Namespace. By relying on external files called schemas, which define the logical meaning and relationships of each of the XML elements, developers build systems decoupling the means of writing and reading the data (XML) from the means of interpretation of the data (schemas). Developers can focus on critical near-term projects knowing that in the future they will gain the benefits of rich semantic interpretation systems, like natural language query or dynamic views of data based on user-defined preferences.

Today XML implies a way of scanning a document via the Document Object Model. This is based on the physical structure of the document. RDF over XML allows one to scan the document based on its more logical structure. Specifically, RDF allows complementary views of data through graphs and nodes, rather than through structured trees, as current XML technology does. RDF will work together with XML-Data schemas to provide a standard way for developers to create relationships for broad classes of XML elements. The trees and nodes used in XML perform a function similar to the tables used in databases. RDF enhances this, by using graphing to relate one XML object to another. Graphing is a very generalized way to represent certain data relationships.

XML Tool Support

Tools offering generalized XML support are available from many vendors, such as ArborText (http://www.arbortext.com/), Inso (http://www.inso.com/), POET (http://www.poet.com/), and Microstar (http://www.microstar.com/), for authoring, editing, storage and DTD generation. Other vendors, such as DataChannel Inc. (http://www.datachannel.com/), UserLand Software (http://www.scripting.com/frontier5/xml), and Vignette Corp. (http://www.vignette.com/) have products available today based on XML for database publishing, content management, and data management. Allaire Corp., ExperTelligence Inc., InterMax Solutions Inc., Pictorius Inc., Powersoft Corp., and SoftQuad Inc. recently committed to providing XML support in their products in early 1998.

Microsoft expects a wide variety of other middleware applications to be developed in the coming months that translate information currently stored in databases into XML for delivery to the desktop. In addition, Microsoft expects rich authoring, schema design, and application developer tools to support XML as well as for databases to store and emit XML directly. Data format-specific tools such as wizards will need to be developed as new vocabularies are defined.

Like Web development tools, there will be XML tools suited for authors as well as developers. The programming tools generally take the form of visualization tools and software code libraries that authors can use to create and manipulate XML content. Software libraries usually come first. For example, Internet Explorer 4.0 includes an XML object model, which can be used from C, Java or scripts, that tools developers can build on to create high-level XML visualization tools or other XML development tools.

Microsoft and other companies are working with the W3C to develop an XML Object Model standard. In addition, the Microsoft XML Parser in Java is available for download, and this site includes documentation on the XML Object Model used in Internet Explorer 4.0 for manipulating CDF and OSD files. Equipped with the ability to parse XML documents, programmers can start building high-level tools that enable authors (and users) to create, edit, browse and search XML

R-789

documents. These tools range from general-purpose editors conversant in any XML vocabulary to vocabulary-specific applications.

Broad industry support for generalized XML and specific formats will quickly make XML tools as ubiquitous as HTML.

In the future, many application categories such as databases, messaging, collaboration, and productivity applications will incorporate support for new XML vocabularies as they are defined. This will enable interoperability within an application category, as well as across application categories, to allow address information in a customer database to be easily shared with a PIM application or e-mail client.

Summary

Extensible Markup Language is flexible enough to allow representation of a wide range of data, and it also allows this data to be self-describing so that structured data expressed in XML may be manipulated by software that doesn't have any built-in description of the meaning behind the data. XML provides a file format for representing data and a schema for data to include a description of its own structure. Microsoft supports XML today in Internet Explorer 4.0 and has plans to support XML as an enabling technology in numerous products. With its powerful expressiveness and flexibility, XML promises to add structure to data on the Internet over standard HTTP, bringing the Web one step closer to realizing the potential for universal communication with anyone, anywhere.

R-790

Exhibit R. Bosworth, General Manager, Microsoft Corp., Europe '98, Microsoft's Vision for XML (May 18-21, 1998) viewed at http://xml.coverpages.org/ bosworthXML98.html on July 21, 2007, 10 pp.

Europe '98

Microsoft's Vision for XML

SGML/XML '98, Paris
From the Opening Keynote Address
Adam Bosworth, General Manager, Microsoft Corporation

See also the presentation slides.

Summary

The premise of this memo is that with the advent of the web, it is now possible to design a truly simple, flexible and open architecture that allows all applications on all machines on all platforms to interact. A second key premise is that this will only be possible through the aegis of logical views where, conceptually, the applications are interacting through agreed upon logical conventions, rather than directly upon either the database tables or methods of the other. The argument is that XML is the key building block for such architecture and that several key XML grammars and conventions will be required for this revolution to be realized. The rest of this paper spells out the building blocks Microsoft believes are required.

The Vision

All applications on the web are easy to make open. Goods and services are easy to find. Any customer can:

- Discover all sites that have some used book he/she cannot find. Then order the book from one of them.
- Move to the country for a month and easily discover who can cut his/her lawn on Saturdays. Then book someone to do so.
- Open a spreadsheet or his/her own custom pages or Java applications and easily let any of them talk directly to any site that manages the customer's portfolios. Then make changes to the portfolio.

In short, make it easy to discover and interact with structured data and applications on the web.

Arguments

Whenever one thinks about a new architecture, one should consider what works and what does not. If the architecture is a sweeping one such as one required to allow applications to interact across the web, then reasonable places to learn lessons are the success of the web itself and the lessons we have learned about servers when used in applications.

R-792

So, what have we learned from the Web?

First and foremost, we have learned that it isn't enough for something to be possible. It must be easy, open and flexible. The web predates HTML of course, but until the advent of HTTP and HTML, it didn't really explode. Why? The answer, succinctly, is empowerment. Once HTML and HTTP arrived, more people could play more easily. The solutions were not necessarily optimal from the point of view of performance or even robustness. They were optimal from the point of view of ease of getting started. In short, they were drop-dead simple. Many people point out the deficiencies of HTML, especially because of the sloppiness of its grammar.

This is true and even regrettable, but the fact is that the simplicity of the HTML model where tags were used not abstractly despite what every book preached, but concretely to describe intended look and feel made HTML immediately approachable. The fact that HTML simply ignored unknown tags made it easy as well since mistakes were silently ignored. Now, as we all know, the lack of formality led to a mess in which few can implement an HTML engine since it is expected to maintain perfect visual and scripting fidelity with what is, essentially, an unwritten, arbitrary, and complex standard. We should also learn from this and keep the simplicity without the mess. Nevertheless, the key point is that HTML exploded because it crossed some threshold of cognitive simplicity. The lesson is that simplicity and flexibility beat optimization and power in a world where connectivity is key.

There is a second lesson which is key. Applications need to be constructed out of coarse-grained components that can be dynamically loaded rather than single large monolithic blocks. In the HTML world, these components are pages. In the applications world in general, however, this lesson applies. The reason for this is simple. The application starts more quickly, only consumes the resources it really needs, and most importantly can be dynamically loaded off of the net. Why is this so important? It is important because of deployment. Applications that can be dynamically loaded in from a central place don't require some massive, complex and difficult installation process onto clients' machines. Note that Java per se doesn't give one this. It is easy, as anyone who has built a large and complex Java application can testify, to build one, which requires literally hundreds of classes to run. That is monolithic. HTML had the serendipitous effect of forcing application designs to partition the application. To repeat, the lesson is that applications should be loaded in coarse-grained chunks.

What have we learned from servers?

We use a simple analogy here. The analogy is the corner grocery store. Imagine that there is such a neighborhood store and everyone buys his or her weekly groceries there. Now imagine that all of a sudden, everyone's buying patterns changed. Instead of buying a few days supplies at a time, they bought a single item at a time. In short, a customer would come in, buy a quart of milk and leave. Then return, buy a stick of butter and leave. Then return, buy a bag of apples and leave. And so on. In short order, there would be huge lines trying to buy their groceries at the store. If each time, the customers were buying their goods using electronic cards or checks (we know, only in backward USA), it would be even worse. Conversely, suppose that all panicked about coming inflation and came in and tried to buy a year's supply of groceries at once to bring home and put into 17 freezers. An entirely new and different set of problems would surface. Happily customers don't do this.

It turns out that servers are like grocery stores. They can only handle so many customers at once (check out counters). More customers mean longer queues whether at a grocery store or on a server. They cannot scale beyond a certain point (you cannot build checkout counters on the fly) and servers cannot just run more and more processes concurrently. They start competing for the CPU and overall performance actually degrades.

R-793

The cost of starting a new conversation with each customer (client) or finishing it has very real costs and if the transactions are too fine-grained, these costs can dwarf the other costs. Servers aren't designed to serve up really huge amounts of data (a year's supply of groceries) either. They bottleneck, TCP/IP complains, and so on. We also learned that processes shouldn't hang onto state while waiting on clients or other slow processes. As an example, if the clerk doesn't know the price of an item, it would be unfortunate if the clerk simply stopped until the price were found and went to find the price out. The whole queue would block. Instead, the clerk should get someone to find out the price and even start helping the next customer if the delay will be significant. In server-speak this means that the process shouldn't hang onto state. What we learned from servers was to conduct coarse-grained interruptible conversations with them.

This is very important. It means that when talking to a medical system or a payments system of your bank or a purchase order system, the client should exchange information in coarse-grained granularity; chunks large enough to let the client go away and do useful work for a spell. Indeed, often the process should dump all its state up to the client and grab it on the way back just to allow the process to quickly take the next task off of the queue. In short, it means that good application design for talking to applications on servers involves relatively few methods that can accept and return complex rich sets of information such as the complete patient record or the complete portfolio description or the complete purchase order. This is not how we normally design methods where we explicitly design for encapsulation and thus make every access to every property or element a call.

There is another key lesson to be learned from servers. The code on the server is designed to talk efficiently and swiftly to share transacted resources like databases or payment authorization systems or airline reservation systems. It is code that fundamentally assumes that it is connected through distributed transactions to these resources. This in turn implies rapid turn-around. Why? Because while any process is holding a transaction on resources, they are locked and this hurts other clients' ability to even read such resources. This code cannot move to a place that isn't similarly connected to these resources. It would lock up these servers. If a banking transaction, for example, is debiting an account from one system and crediting an account from another, all inside of a distributed transaction, it must not run over slow or unreliable lines because it will start to lock up the resources. This means that the code that deals with this information on the client isn't the same code. It shares the same information (purchase orders, portfolios, personnel records), but it is code dedicated to some different process such as letting the user view the information. The lesson is, in short, to move the information and possibly some simple portable validation logic for the information, but not the custom code that handles this information on the server. We tend to refer to this as an object to object bridge. Which is, of course, an RPC. So, to repeat, the lesson to be learned is to share data, not code.

To summarize, we can learn the following lessons:

- Simplicity and flexibility beat optimization and power when connectivity is key
- Applications should be loaded in coarse-grained chunks
- What we learned from servers was to conduct coarse-grained interruptible conversations with them
- The lesson to be learned is to share data, not code

Microsoft's Opinion

XML is tailor made as a way to move this coarse-grained information around the web, whether the information is purchase orders, personnel records, portfolios, or just parameters. It is open, easy, and flexible. It is self-describing. If the proposals in XML-DATA are adopted, it supports rich extensible

R-794

data-types and additional meta-data that a tool might want to understand how to use or interpret this data. Virtually any logical view's data can be described and transported using XML although XML does prove to be cumbersome for expressions and script and could use some enhancement particularly in the latter area.

What is left to do to enable this interaction?

Enhance XML to support extensible and rich typing and meta-data:

The logical views that describe the data that applications will deliver or accept must include a rich and extensible model for data types. As soon as one tries to use XML even for something really simple like transporting the arguments that an RPC implies or transporting a simple set of rows from a relational database, one discovers the need for data types. The recipient needs both to be able to discover the data types and to be able to reliably parse them into the appropriate types for each programming language. Furthermore, the negotiation process will often involve the conveyance of other meta-data important for the recipient. For example, if I am shipping around task descriptions as part of a project description, I may have a default Java class or COM class that should be materialized from this information. If I am a middle-ware search engine, I may want to get enough meta-data about the elements to know which should be shown in a summary view and which elements should be used as links to related information and how. In short, the amount of ancillary information that might be desired for XML data is essentially unlimited and this argues for a model for describing the schema of an XML grammar that is extensible. All of this is, of course, exactly what the XML-DATA proposal is intended to address.

Describe the grammars:

In order to illustrate this section, we ask the reader to imagine a scenario in which the customer is trying to search for providers of used books. The customer has his/her heart dearly set upon some out of print tome by Thomas Carlyle and wants to know which sites have this book. The eager entrepreneur has decided to deliver such a service and has managed to convince many used book providers and even some online book retailers to publish their inventory using a particular schema to describe books. It need not even be a particularly good or comprehensive schema for describing books.

Remember that sites can support multiple schemas (logical views) against a single implementation. The enterprising entrepreneur has also convinced Yahoo to point to the site for book searching which lends certain urgency to the book purveyors to actually publish this schema. Now comes the need for grammars. Which ones?

First, of course, a grammar for the books themselves and most likely a grammar for a site to describe its physical ability to deliver goods and take payments. XML can do this today.

Second, however, is a way for a search engine to quickly and efficiently determine for each site whether that site has any URL that returns the desired schema. This requires some grammar and some protocol or convention. The grammar is one that enumerates which URLs return which schema (if any). The protocol or convention is either a magic URL or an HTTP header or verb. This is used to make it clear to the site that the desired information is the list of resources with the filtering restrictions if any (e.g. which resources support the book schema) that are being passed in through a particular XML grammar. Thus we are likely to need two XML grammars, one to describe a filter, and one to present a logical

R-795

view of a site in terms of URLs and schema that URLs can return. Let's call this latter one "Site description" and the former one "Filters" for the moment.

Third, let's assume that the search engine has a way to now discover which URL's return book schema. Now the search engine may do one of two things:

1. Pull down the inventory and merge it into a giant database that the search engine stores tracking which sites have which books. In this case the search engine would like some additional information. It would like to know if the book-purveying site would let the search engine "subscribe" to this URL. In this case, the search engine is likely to want to "subscribe" to the site. This would mean that, from time to time, the site would be told that the inventory on hand has changed. The site would then either ask for the entire inventory over again or ask for essentially a DIFF to the inventory already on file. If the site did the latter we would need yet another XML grammar to describe changes to make. We'll call this grammar "Updates". Presumably the search engine would discover whether the book-purveying site was willing to accept subscriptions to the URL through the same "Site description" grammar that described the URL in the first place.

2. Remember that the site has books. In this case as well, the site would like to find out some additional information. It would like to ask whether the book purveyor site was willing to accept "Filter" grammar requests asking for filtered sets of these books. Then, if a request for books came in, the search engine would ship such a request directly to the book-purveying site using the "Filter" grammar. This model is slower of course, but is a lot easier to implement and less likely to fail.

In either case, as we hope this example shows, standard XML grammars for describing sites, for asking for filtered subsets and for sending changes are likely to be extraordinarily useful.

However, this model may sometimes be too simplistic. Let's imagine that the site isn't willing to support a general query language no matter how limited. What it is willing to do is expose certain URLs which when sent the appropriate parameters of Title or Author return a list of books for those Titles or Authors or both. Notice that this is basically RPC. Now, even in this case, the search engine would want to send the parameters to the book-purveying site in a simple, easy to engineer manner. If the "Site description" grammar assumed above describes the desired shape of the parameters of the "method" and there is a standard grammar for marshalling parameters for RPC or the "Site description" grammar describes the grammar of the search request, then the search engine knows what XML to send. Ideally, this grammar for RPC will also describe how synchronously the results may be returned and by what mechanism to allow for delayed return.

It is interesting to note that the "Site description" grammar could be built as an extension of the current XML-DATA proposal.

There is another standard grammar that shows great promise. This is XSL or Extensible Stylesheet Language. The search engine now has lists of books and this is good. But there is another challenge that faces the search engine: how does it handle the case where different sites support different book schemas? Suppose that there are several competing schemas. It would like to know them all and be able to convert between them into whichever common one it prefers. What would be ideal is an XML grammar that described the logic required to quickly and efficiently convert from one schema to another. XSL has the potential to provide a standard mechanism for such conversions.

Similarly, the client receiving this book list information has a challenge. How does it dynamically

R-796

generate the HTML that would be required to display the book list that vendors might want to include as links back to their sites, little blurbs about their site, and so on? The client could of course take direct advantage of DHTML and JavaScript or Java or any COM language to solve this problem writing either script or a component to custom generate the appropriate DHTML from the data. The client could also use the data binding features of DHTML. But it would be handy to have a *standard* way to describe how to build the HTML from the XML and a standard component (converter) that knew how to execute such instructions. Again, XSL can provide such a conversion.

Lastly, the site vending books has a problem. How does it map its book database, undoubtedly stored typically in a large relational database, into the appropriate XML logical view? Interestingly, the XML-Schema for the view could itself contain suitable meta-data to help one compute the answer to this question, but then a standard for such meta-data decoration of XML-Schemas would be required. Even so, undoubtedly a "plan" would be computed for actually building the appropriate XML logical view and this plan would require an XML grammar, call it the "Database conversion". Also required would be a general converter that given such a grammar, would actually talk to the database, submit the requisite SQL, and build the appropriate XML. This model would be extremely useful on any server mapping relational data to HTML clients, even sometimes building the HTML directly.

So, to summarize, the following XML grammars would be necessary:

1. The grammars proposed in the XML-Data proposal
2. "Site description" grammar: This should probably be either a specific proposal or standard meta-data extensions to the XML-DATA proposal used to describe the "methods" of a site and the "schemas" it returns. Extensions to this model would also be needed to determine for any of these returned schemas, whether the site is also willing to accept the "Filters" or the "Updates" grammar.
3. "Updates" grammar: An XML grammar to describe changes to/from cached XML
4. "Filters" grammar: An XML grammar to describe desired subsets of a particular XML logical view
5. Either an XML RPC proposal or extensions to the XML-IDL proposal above to describe the grammar of parameters submitted in an RPC and requisite envelope information

The following grammars would be useful:

1. XSL: It would be helpful to pay some attention to ensuring that the part of XSL that is a tree transformation language is sufficiently easy and powerful
2. More standard meta-data for XML-DATA to describe mappings of elements to relational stores
3. "Database conversion" grammar: A grammar to construct logical XML views from relational databases

Ensure that the programming models for XML stay simple and appropriate:

R-797

Any model for moving information around the net will involve the requirement that two types of programmers can access this information, programmers using serious structured languages such as Java or C++ and programmers using scripting languages such as JavaScript. It is important to note that the requirements are not identical. The scripting programmer tends to not have either pointers or types and to want to treat XML as a single big tree of nodes which can be navigated through a collection syntax, something like root[foos[3][bars[2][sams[1]]] where this completely fictitious syntax would mean that the script writer wanted the first node with tagname "sams" within the second node with tagname "bars" within the third node with tagname "foos" within the root. And do to the magic of overloading of scripting languages, such a model can be surfaced without requiring each and every XML provider to constantly provide and maintain such indexed collections. What this code would really translate to is code that goes and finds such a node using simple enumerators. Conversely, the C++ or Java programmer will probably happily deal with a lightweight enumeration model and build their own object models on top of it rather than depend upon the implementation vagaries of layers provided by the provider. Indeed, in many cases, the C++ or Java programmer may simply want the nodes pushed into them as they will be building their own data structures. It is important that we not overburden each implementation with some top-heavy API which is neither fish nor fowl, but rather build the right low level API on top of which all can build.

What will be required in addition to simple enumeration are ways to get data based upon the types described in the schema, to make changes to the data or tree, and ways to validate that the changes conform to the schema of the XML document. Lastly, as XML grammars described above emerge, services to automatically execute them will start to be expected as a service layer on top of the base API, but with a model that allows the implementers to support them directly and natively. In other words, while a service layer might be necessary to find some node using simple predicates, a provider might natively implement support for this in a more efficient manner using internal indexes or hash tables or what have you.

Stores:

Many have asked about XML "stores". It is unlikely that there will be one XML store. Different stores will have different purposes.

The cheapest store, of course, is the file system. Many standard components are springing up to provide DOM (Document Object Model) API access to XML stored in streams or files including, of course, our own component which we will make available ubiquitously on all Windows platforms, on Unix, and in Java.

Relational stores are superb at exposing multiple different logical views on the same data and by now have very good scaling and transactional characteristics. Typically most mission critical data will live in them. But this doesn't mean that there cannot be "converters" between the relational stores and XML logical views. We expect to see this as a rapidly exploding part of the XML industry. Ultimately, we would expect that the database stores themselves would be able to make these conversions happen, but that in the near term these converters would be part of middle-ware living on middle tier servers. This is too complicated a subject to describe in depth in this memo.

Object oriented stores. Many object oriented stores have discovered new and additional purposes acting as "staging" stores for XML data as it is cached on the middle tier or the client. Some will undoubtedly turn out to do an excellent job of providing XML caching. What will be critical here is that we avoid a profusion of APIs for talking to these caches if we want to see interoperability. The DOM helps here by describing how to talk to a particular XML document/object, but doesn't handle the larger issues that

R-798

caches are likely to worry about such as versioning, transactions, searching, and so on. We hope that WebDAV will play a major role in driving convergence in this arena. We also expect the object-oriented companies and innovative companies like Frontier to play key roles in this arena.

Converters architecture . . .

Clearly, most data will not be stored in XML. Indeed, the main thrust of the theme of this paper is that XML acts as a convenient mechanism for interacting at the logical view level, not the physical level. Some data will come from relational databases. Some from Teletype feeds. Some from mainframe databases through CICS. Some from SGML. Some from object-oriented databases. Some from mail stores. Much will be synthesized dynamically by objects written by programmers.

But if we want a model for interacting, we will need a standard component model for transforming between any data and XML. We call such components converters.

A normal middle tier server will have mechanisms for wiring such converters up to queues of XML messages. For example, clients may be sending requests for available classes to a university and expecting a specific logical view (XML schema). The server at the university would pop such requests off of a queue, pass the XML request to the appropriate converter, probably hooked up to the class schedule database on the back end, take the resulting XML, and route it back to the requestor. The requestor would then display the information, review it, and then, perhaps, want to enroll. The requestor would send a message with another schema (enroll) to the middle tier. The middle tier would pop this message off of some queue and quite possibly hand it off to a completely different converter component talking to a backend student database.

We expect to help produce three sorts of converters:

Objects written in code:

This requires a component model for being an XML converter. A converter will need to act a lot like any other XML provider. It will deliver up XML on request. It will support some standard discovery mechanism such as the "Site discovery" grammar. It will need to be able to stream results out. We need to make it easy for people to build such components, even using scripting languages.

Database converters:

An extraordinarily common case will be mappings between XML logical views and SQL databases. This is sufficiently common that it should be possible to author such transforms without having to write code. Indeed, it should be possible to simply send a request for a particular schema described using the XML-Schema syntax defined in the XML-DATA proposal, but augmented with information required to map the elements to relational database elements.

XSL (Extensible Style-sheet Language):

XSL has been traditionally viewed as a component that took XML in and emitted nice printed output using whatever print-medium was chosen. We have looked at XSL somewhat differently. Our chosen output medium is DHTML, whether for dynamic interactive output or for producing rich Office documents. However, we don't want to require that everyone who is trying to produce user interface for XML be a programmer. We believe that it should be easy to produce DHTML from XML.

R-799

Furthermore, we also don't believe that everyone will always agree about schema. It is possible, for example, that 3 or 4 or 5 schemas will be used by sites providing information about their book inventory rather than one. However, any sensible program accessing or viewing such data will want to pick one. This will require a mechanism for converting from one XML schema to another. Sometimes, if sufficient semantic information were available, it might be possible to do this automatically.

But frequently such information will not be available and a customer will have to describe how to convert. Again, however, we don't want to limit this to programmers. We believe it should be easy to produce XML from XML.

We want a standard extensible grammar, therefore, for converting from XML trees into either other XML trees or into DHTML trees. Thus to Microsoft, the part of XSL that is interesting is the part that helps define how a given XML tree should be translated into a quite different tree. We see the emergence of standard converters which use XSL messages to decide how to translate from one XML grammar into another or into DHTML.

Implications of all of this for the web:

XML is the building block. XML-DATA is the required next step. Then, as we have seen, we do need agreement upon some specific XML grammars and we must remember to keep these grammars simple, open, and easy. If we do all this, the power we unleash to the normal developer is incalculable. It becomes possible to build systems for any line of business application, for collaboration, and systems for intelligent information retrieval including for goods and services.

This revolution will do for applications what SQL partially did for databases. It will open up the floodgates because the number of people who can interact with them will increase enormously. Many hard issues remain to be worked out including the details of these grammars, the security issues, transactions, and so on. But in the spirit of the web, we have submitted our proposals for most of these pieces to allow us to start and learn rather than trying to get it perfect and never shipping.

This will cause a site revolution. Well behaved sites will support not just *data* for specific schema on request, but *filtered sets of data*, *updates to data (notifications)*, and requests to update their data (data entry) and, in all these cases, they will support these services using *standard* XML grammars. It will open up an entire new business, namely converters with a myriad of specialized XML grammars that make sense for particular types of transformations.

Predictions:

The requisite grammars will be defined and implemented. The DOM will converge and streamline. Many database storage systems will start to support both the grammars and the DOM. XML will become a widespread solution to interoperable RPC on the net. Microsoft will actively support all of the above.

Tools will emerge to:

- help author the grammars (Schema, Filters, Updates)
- view, edit, and manage XML
- define mappings between XML logical views and databases
- help programmers use all these components and automatically inter-operate with these

R-800

components.

Server components will emerge to:

- act as stores and managers and queues for XML
- act as converters between XML and relational databases
- act as converters between XML and Schema
- marshall objects directly from XML

The programming model for building applications will change to:

- One in which XML is the standard message format used for transmitting data and for RPC. There will be standard ways to manage these messages through queues and to register events for handling these messages. Components on the server will routinely save their state into XML data-stores virtually transparently.

Summary

We're only at the *very* beginning of the Net revolution. The most exciting part is still to come. Soon it will become as easy to interact with programs and data all over the net as it currently is to view shared presentation and content.

R-801

Exhibit S. Winer, XML-RPC for Newbies (July 14, 1998) viewed at http://www.scripting.com/davenet/1998/07/14/xmlRpcForNebies.html on July 21, 2007, 6 pp.

XML-RPC for Newbies

Prev
Next
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
About

Tuesday, July 14, 1998 by Dave Winer.

Yesterday I sent you a pointer to a story about Microsoft and XML-RPC, which according to Microsoft, is in the early stages of development.

A quote from the piece: 'Microsoft officials said the XML-based protocol fits into its goal to interoperate with other platforms and vendors. "We're committed to interoperate as never before," said Vic Gundotra, director of platform marketing at Microsoft.'

Right on! It's taken fifteen years for this issue to rise to the top as an important agenda item for the software industry. I hope it stays right at the top until we get a solid Internet-based cross-platform remote procedure calling protocol.

But first... What is it?

XML-RPC for Newbies

Inside every computer, every time you click a key or the mouse, thousands of "procedure calls" are spawned, analyzing, computing and then acting on your gestures.

For example, when you move your mouse over an icon, the computer calls a procedure, LocateMouse, to figure out what you're pointing at.

Is the mouse pointing at a menu? At a scroll bar? If so, which part of the scroll bar? Is it pointing at an icon? Or some text? Every possibility is considered.

OK, so let's say the mouse is pointing at an icon when you click the mouse button. What kind of icon is it? If it's a printer, call the procedure that prints things. What do you do when the user clicks the mouse on this kind of icon? A special procedure answers that question.

This is the kind of chatter that's going on inside your computer *all the time*, even when you aren't there. It's always asking questions. And the answers come from procedures. To get an answer, the software "calls" the procedure.

Parameters and returned values

R-803

Along with the call, the procedure might require some extra information, so it doesn't have to recompute things that other procedures might have already figured out. These are called "parameters".

The LocateMouse procedure might need to be told where the mouse is. The location of the mouse is usually expressed in a coordinate system like the Cartesian plane that you learned about in high school. An x coordinate and a y coordinate. Such a procedure would be said to "take" two parameters, an x and a y.

Parameters are important for three reasons. First, why do the work again when the caller probably already knows where the mouse is? And second, the mouse may have moved in the time it took to call the LocateMouse procedure. And third, a procedure may be called to do some computation, for example, to look up a record in a database. Such a procedure would require a user's name or account number as a parameter, to identify the record that's to be looked up.

And there's the motivation for the third part of a procedure call -- the returned value. It's the answer that the procedure sends back to the procedure that called it. A database-access procedure call might return a set of values, all the elements of the record indicated by the key identifier it received as a parameter.

What is a procedure call?

So, now we're ready to say, concisely, what a procedure call is.

A procedure call is the name of a procedure, its parameters, and the result it returns.

Why are procedure calls important? A very simple answer. Without them, there would be no computers!

Every program is just a single procedure called main, every operating system has a main procedure called a kernel. There's a top level to every program that sits in a loop waiting for something to happen and then distributes control to a hierarchy of procedures that respond. This is at the heart of interactivity and networking, it's at the heart of software.

What is RPC?

RPC is a very simple extension to the procedure call idea, it says let's create connections between procedures that are

R-804

running in different applications, or on different machines.

Conceptually, there's no difference between a local procedure call and a remote one, but they are implemented differently, perform differently (RPC is much slower) and therefore are used for different things.

Remote calls are "marshalled" into a format that can be understood on the other side of the connection. As long as two machines agree on a format, they can talk to each other. That's why Windows machines can be networked with other Windows machines, and Macs can talk to Macs, etc. The value in a standardized cross-platform approach for RPC is that it allows Unix machines to talk to Windows machines and vice versa.

What is XML-RPC?

There are an almost infinite number of formats possible. One possible format is XML, a new language that both humans and computers can read. XML-RPC uses XML as the marshalling format. It allows Macs to easily make procedure calls to software running on Windows machines and BeOS machines, as well as all flavors of Unix and Java, and IBM mainframes, and PDAs and sewing machines (they have computers in them too these days).

With XML it's easy to see what it's doing, and it's also relatively easy to marshall the internal procedure call format into a remote format.

Why RPC is important

OK, now that we understand what XML-RPC is, let the XML part fade into the background. It's an implementation detail. Programmers are interested in XML, as are web developers, but if you're a user or an investor, XML is about as important as C++ or Java. The developers like it, or seem to, and that's the only major take-away from the XML part of XML-RPC.

But RPC is important, no matter what format is used, because it allows choices, you can replace a component with another one; and it opens possibilities, empowering advanced users to develop solutions with packaged software that the developers didn't anticipate.

Choices

For me, the trail goes back to the mid-80s when my company released an integrated product called MORE which was both

R-805

an outliner and a presentation product. There was a lot of synergy in the integration, but a significant number of users just wanted the (new) presentation features without the outliner; and a significant number wanted the well established outlining features, and had no interest in presentation features.

Both groups wanted a choice, they wanted us to separate the two pieces and connect them together. If someone wanted the outliner they wouldn't have to buy the presentation software and vice versa. But if they had both, you'd get all the integration features. It was a great idea! I've told this story many times. It was a turning point in my career. The users were asking for something reasonable, yet there were no operating system underpinnings to allow such a separation.

A few years later, the needed technology was in the operating system, and we had built software, a database and scripting engine, around the technology. Application developers implemented interfaces that allowed our software to "call" them. We'd send some parameters to a procedure, implemented in another app, and the app would send back an answer. Every app that implemented this could now be viewed as a toolkit that an advanced user could build new applications thru. In this world, we could have done an outliner and a presentation program, and linked them together with "glue" scripts.

Choice is good for users, and for some developers, in some situations, it's not that good. So it was a mixed bag. In some cases, web servers for example, there were absolute standards that were adhered to. And in others, page layout programs for example, the "wires" were seen as a competitive advantage, and never got in synch.

Possibilities

Another advantage of RPC is that users may see opportunities for integration that developers don't understand, combining the features of two or more products. With the ability to glue apps together, an inspired user can do it. The developers may never understand or care what the users are doing, but it can happen anyway. Interfaces provide room for growth, open doors, allow a user-perspective to create new functionality. This vision says that users can be empowered to make integration choices without the support or help from software developers.

This vision was realized in some areas. Databases and web sites are a hot area today, but all the pioneering work was done by users! How many database vendors understood the

R-806

web when it first started booming? But flip the question around, how many web developers needed databases? Almost all of them figured it out.

Because there are standard interfaces for databases, the web people could do what they wanted, even though the database developers didn't have a clue what they were doing.

Trade-offs

As with all software technologies, there are trade-offs. When you integrate, when the procedure call is within the same app, you get much greater performance and more user-oriented features because it's easier to do a procedure call within a single app than it is to go outside of the app with a remote procedure call.

Apps will not disappear. In the early heady days of this sub-industry, some people thought they would. But there's too much momentum behind the idea of a process implementing internal procedure calls. So, long live the app! It's a good thing. It's still here, for a while at least.

Back to Microsoft

According to Jeff Walsh's InfoWorld article, Microsoft is planning to open up their operating system using XML-RPC. Such a protocol could be deployed quickly in other operating systems that support HTTP, ranging from Perl scripts running on Linux, to Quark publishing systems running on Macs, to relational databases running on mainframe systems. It could put Windows at the center of a new kind of web, one built of logic, storing objects and methods, not just pages and graphics.

As you know, I'm a big believer in this kind of glue. I've been talking with Bill Gates about this since 1985, when we met to talk about interapplication communication for MS-DOS. Then in 1991 we revisited the idea as Apple was deploying the Apple Event Manager on the Macintosh OS. Apple wasn't willing to license the technology, so Microsoft did COM. But COM mainly runs on Windows, and although there have been recent efforts to popularize it on other operating systems, it'll be an uphill battle because most operating systems have their own native way of connecting systems together at a remote procedure call level.

So, by agreeing, at least at a philosophic level, that XML-RPC is an important way to go, Microsoft is putting out a big Welcome Mat -- come rule our world, they say. You can

R-807

control Windows without adopting COM. You can replace an NT machine with any other machine that supports the same interfaces.

Even if NT should become a locked-in standard in the enterprise (far from an accomplished feat at this time) users will be able to substitute other operating systems for NT thru simple interfaces. I support that, I think it's a winning strategy, because I know how much value customers place on choices and new possibilities.

It's the lesson that the Internet teaches. It's a lesson that Gates learned so well that he's upping the ante. Instead of a hodge-podge of different wire protocols and payload formats, let's get something in place that can quickly be widely deployed everywhere. Why wait? That's what Microsoft is asking.

There's no reason to wait. Let's get just-enough technology in place and then let a thousand flowers bloom. I absolutely don't care what the wire format is, as long as I can develop simple scripting interfaces on top of it. I give up control, everyone does, and we let the messy cacaphony of competition take over.

That's the way to go.

Dave Winer

Updated 7/21/98: The XML-RPC spec is available. DW

R-808

Exhibit T. Walsh, Microsoft spearheads protocol push, InfoWorld Electric (July 10, 1998) viewed at http://infoworld.com/cgi-bin/displayStory.pl?98071.whsoap.htm on July 21, 2007, 2pp.

The page cannot be displayed

The page you are looking for is currently unavailable. The Web site

Microsoft spearheads protocol push

By Jeff Walsh
InfoWorld Electric

Posted at 5:05 PM PT, Jul 10, 1998

Microsoft is quietly working on an early implementation of a new protocol to build distributed computing applications that span the Web.

The Simple Object Access Protocol, or SOAP, enables Remote Procedure Calls (RPCs) to be sent as Extensible Markup Language (XML) syntax across the Web's HTTP architecture.

The protocol was developed by Microsoft, UserLand Software, and DevelopMentor, according to the draft specification of the protocol being circulated by Microsoft.

Microsoft officials said the XML-based protocol fits into its goal to interoperate with other platforms and vendors.

"We're committed to interoperate as never before," said Vic Gundotra, director of platform marketing at Microsoft.

According to those who saw early demos, SOAP bridges Component Object Model (COM) and Distributed COM objects across the Web and runs natively in Windows NT, Windows 95, and Windows 98. Microsoft has also built SOAP connections to Internet Explorer and to Java, sources said.

Microsoft's COM architecture speak natively to the SOAP protocol, Gundotra said. He added that other object architectures, such as CORBA, Java Remote Method Invocation, and Apple Events, could use SOAP as a "lowest common denominator" to interoperate, though he warned this protocol is early in the development cycle.

Using text to transmit objects across the Web holds a lot of promise, said one analyst.

"It's very powerful. If you want to talk about thin clients, you can't get thinner than that," said J.P. Morgenthal, president of NC.Focus, a consultancy in Hewlett, N.Y.

At present, other companies are shipping products that enable RPC over XML, such as UserLand Software's Frontier 5.1 Web content management system, DataChannel's WebBroker technology, and WebMethods' B2B Integration Server. None would talk directly about SOAP, but all liked the idea of a single standard.

Dave Winer, president of UserLand Software, in Palo Alto, Calif., said he would be happy to

R-810

standardize on one protocol.

DataChannel has submitted WebBroker, a similar protocol, to the World Wide Web Consortium. John Tigue, DataChannel's senior software architect, said SOAP will most likely be similar to his submission, but he looks forward to working with Microsoft on one standard.

Josh Lucas, a Web consultant in Boston, is excited about Web-based distributed computing.

"Developers can start piecing together applications and make more meta applications," Lucas said. "A standard like this would go beyond the cross-platform promise of Java."

Microsoft Corp., in Redmond, Wash., can be reached at http://www.microsoft.eom.

Jeff Walsh is a reporter for InfoWorld.

Related articles

"Vendors plan to extend XML's reach"

"DataChannel's Dave Pool talks about shaping the role of XML to suit different needs"

"W3C gives a nod to DataChannel's WebBroker"

Go to the **Week's Top News Stories**

Please direct your comments to InfoWorld Deputy News Editor, Carolyn April

HOME | NEWS | TEST CENTER | OPINIONS | FORUMS | CAREERS | STOCK QUOTE
TECH INDEXES | SUBSCRIBE | ABOUT US | SEARCH

R-811

Exhibit U. Merrick et al., US 7,028,312, XML Remote Procedure Call (XML-RPC)

US007028312B1

(12) United States Patent
Merrick et al.

(10) Patent No.: US 7,028,312 B1
(45) Date of Patent: Apr. 11, 2006

(54) XML REMOTE PROCEDURE CALL (XML-RPC)

(75) Inventors: **Phillip Merrick**, Burke, VA (US); **Stewart Allen**, Fairfax, VA (US); **Joseph Lapp**, Fairfax, VA (US)

(73) Assignee: **webMethods**, Fairfax, VA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/274,979**

(22) Filed: **Mar. 23, 1999**

Related U.S. Application Data

(60) Provisional application No. 60/096,909, filed on Aug. 17, 1998, provisional application No. 60/079,100, filed on Mar. 23, 1998.

(51) **Int. Cl.**
G06F 9/44 (2006.01)

(52) **U.S. Cl.** **719/330**; 709/219; 709/203

(58) **Field of Classification Search** 709/310 320, 709/328, 329, 330, 200–203, 217–219; 719/310–320, 719/328–329, 330
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

6,041,365 A * 3/2000 Kleinerman 709/328
6,466,971 B1 * 10/2002 Humpleman et al. 709/220
6,480,860 B1 * 11/2002 Monday 707/102
6,546,419 B1 * 4/2003 Humpleman et al. 709/223

OTHER PUBLICATIONS

Merrick et al., "Web Interface Definition Language (WIDL)", NOTE-widl-970922, http://www.w3.org/TR/NOTE-widl-970922, Submitted to W3C Sep. 22, 1997, pp. 1-16.*

(No author given), "Extensible Markup Language (XML)—W3C Working Draft Nov 14. 1996", http://www.w3.org/TR/WD-xml-961114.html. Nov. 14, 1996, pp. 1-28.*

Bosak, Jon, "XML, Java, and the future of the Web", Sun Microsystems, http://www.ibiblio.org/pub/sun-info/standards/xml/why/xmlapps.htm, Mar. 10, 1997. pp. 1-9.*

Goldfarb et al., "The XML Handbook", Prentice Hall, pp. 555-568, 1998.*

Winer, Dave, "RPC over HTTP via XML", http://davenet.userland.com/1998/02/27/rpcOverHttpViaXml, Feb. 27, 1998, pp. 1-7.*

(No author given) "Extensible Markup Language (XML)—W3C Working Draft Nov. 14, 1996", W3C, document# WD-xml-961114, pp. 1-27, Nov. 14, 1996.*

(No author given) Extensible Markup Language (XML) 1.0—W3C Recommendation Feb. 10, 1998, document# REC-xml-19980210, p. 1-32, Feb. 10, 1998.*

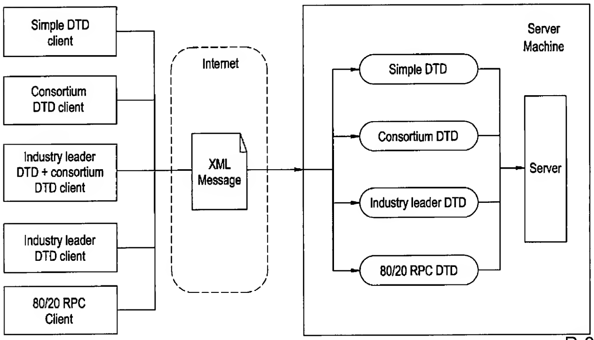
(Continued)

Primary Examiner—St. John Courtenay, III
(74) *Attorney, Agent, or Firm*—Sughrue Mion, PLLC

(57) **ABSTRACT**

A Remote Procedure Call (RPC) uses a message expressed in a mark-up language message encoding, with type labels indicating data item types.

55 Claims, 13 Drawing Sheets



R-813

OTHER PUBLICATIONS

Winer. Dave, "Frontier and XML", http//frontier.userland.com/stories/storyReader$1124, Mar. 1, 1998, p. 1-2.*

St.Laurent et al., "Programming Web Services with XML-RPC", O'Reilly, Forward section, pp. ix-xii. 2001.*

Tigue, "XML Enabled Mechanisms for Distributed Computing on the Web", Documation East 1997, Oct. 20, 1997.

Winer. "RPC over HTTP via XML", http://davenet.userland.com/1998/02/27/rpcOverHttpViaXml, Feb. 27. 1998.

Tigue et al, "WebBroker: Distributed Object Communication on the Web", http://www.w3.org/TR/1998/NOTE-webbroker-19980511/, May 11, 1998.

Walsh, "Microsoft spearheads protocol push", http://www.infoworld.com/cgi-bin/displayStory.pl?980710.whsoap.htm. Jul. 10, 1998.

Winer. "XML-RPC for Newbies", http://davenet.userland.com/1998/07/14/xmlRpcForNewbies, Jul. 14. 1998.

Winer. "XML-RPC for Geeks", http://davenet.userland.com/1998/07/19/xmlRpcForGeeks, Jul. 19, 1998.

"Allaire Announces the Web Distributed Data Exchange (WDDX)", http://www2.allaire.com, Dec. 8, 1998.

Winer, "The Politics of Plumbing", http://davenet.userland.com/1999/02/04/politicsofplumbing, Feb. 4, 1999.

http://www.xmlrpc.com. Oct. 1, 1999.

Lapp, "WIDL", presentation slides from a speech delivered at XML '98 on Nov. 17, 1998.

Lapp, "XML IDL and XML RPC", posting to XML.org web site on Sep. 22, 1998.

webMethods Web Automation Toolkit Adds Visual Basic and "Intelligent Document" Support, webMethods press release Dec. 10, 1997.

"webMethods Announces XML-Based Web Automation Toolkit To Be Available Free on the Internet", webMethods press release Mar. 4, 1998.

"What Is XML?". http://www.webmethods.com/xml/about_xml.html, Feb. 20, 1999.

* cited by examiner

FIG. 1

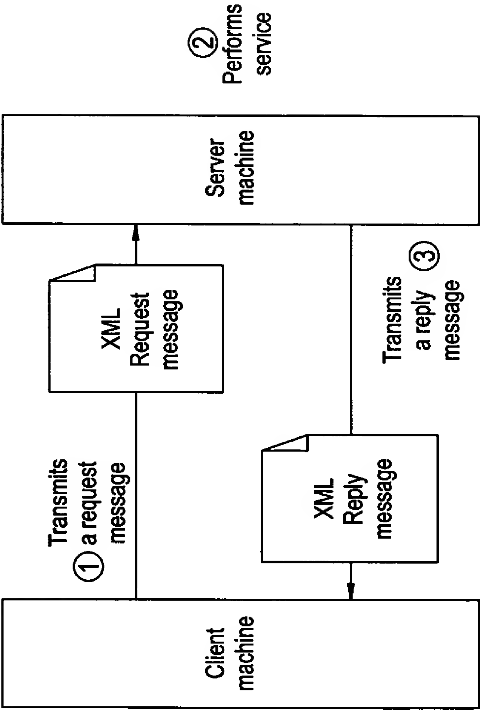
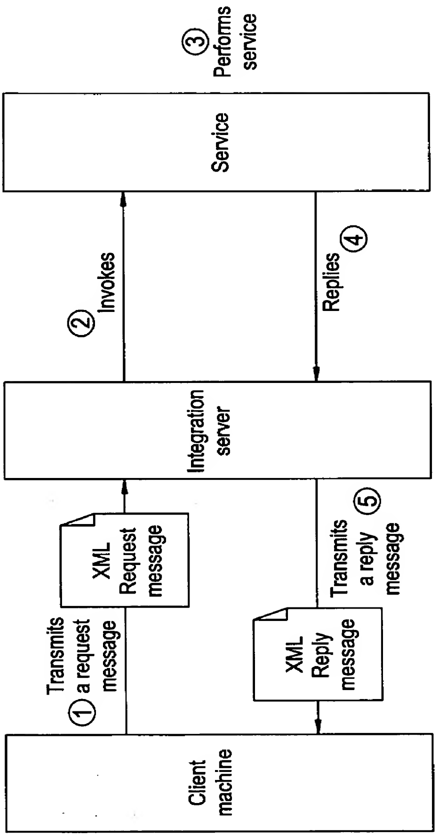
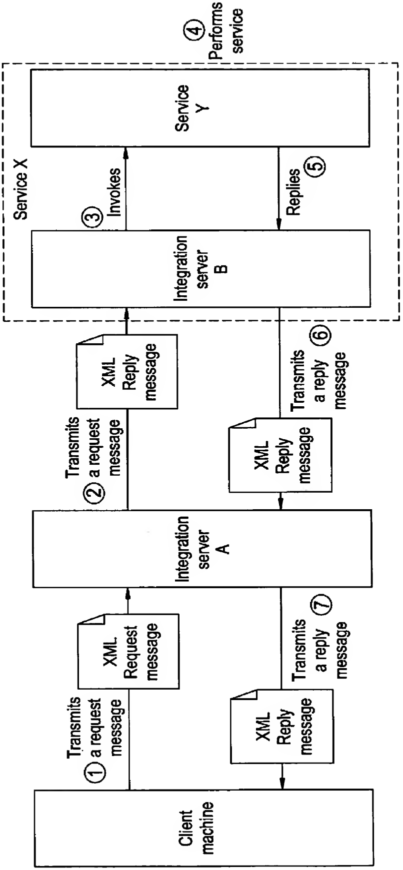

Client machine
1 Transmits a request message
XML Request message
Server machine
2 Performs service
3 Transmits a reply message
XML Reply message

FIG. 2


Client machine
1 Transmits a request message
XML Request message
Integration server
2 Invokes
Service
3 Performs service
4 Replies
5 Transmits a reply message
XML Reply message

R-816

FIG. 3



Client machine
1 Transmits a request message
XML Request message
Integration server A
2 Transmits a request message
XML Reply message
Service X
Integration server B
3 Invokes
Service Y
4 Performs service
Replies 5
Transmits a reply message 6
XML Reply message
Transmits a reply message 7
XML Reply message

R-817

FIG. 4

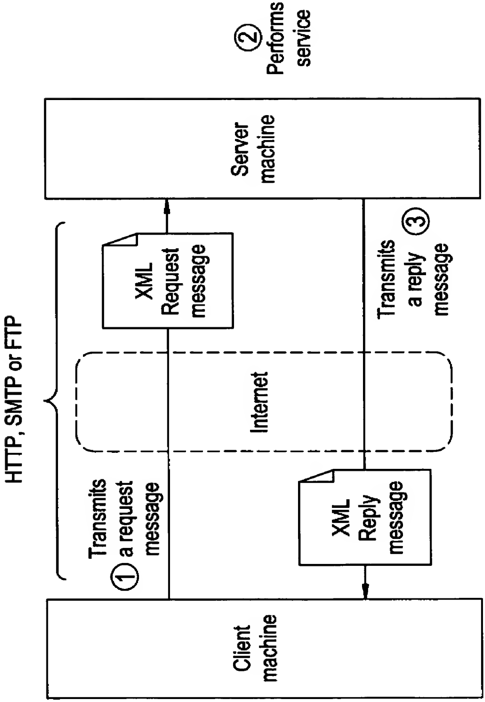

HTTP, SMTP or FTP
Client machine
Server machine
Internet
1 Transmits a request message
XML Request message
2 Performs service
3 Transmits a reply message
XML Reply message

FIG. 5

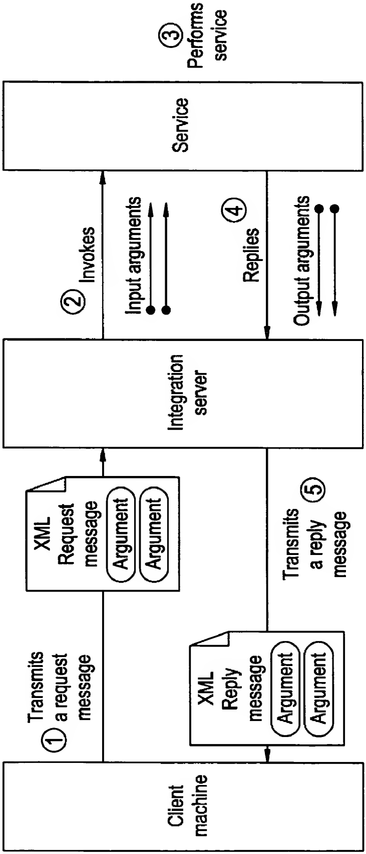
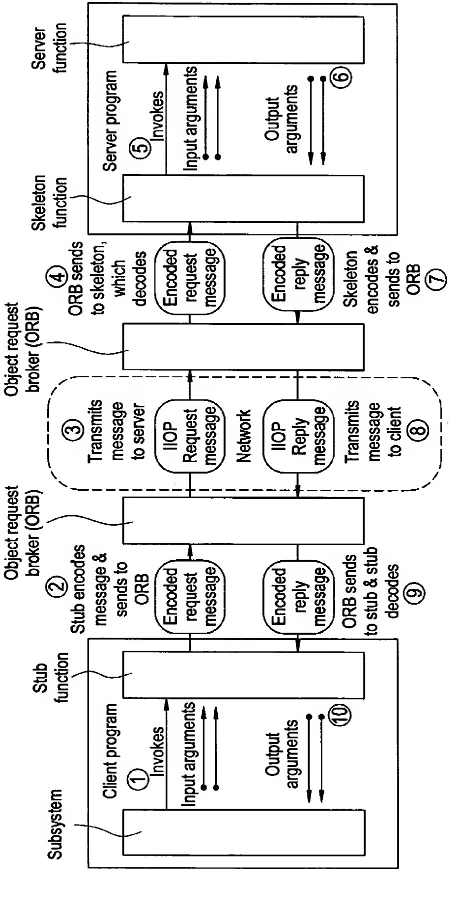

Client machine
1 Transmits a request message
XML Request message
Argument
Argument
Integration server
2 Invokes
Input arguments
Service
3 Performs service
4 Replies
Output arguments
Transmits a reply message 5
XML Reply message
Argument
Argument

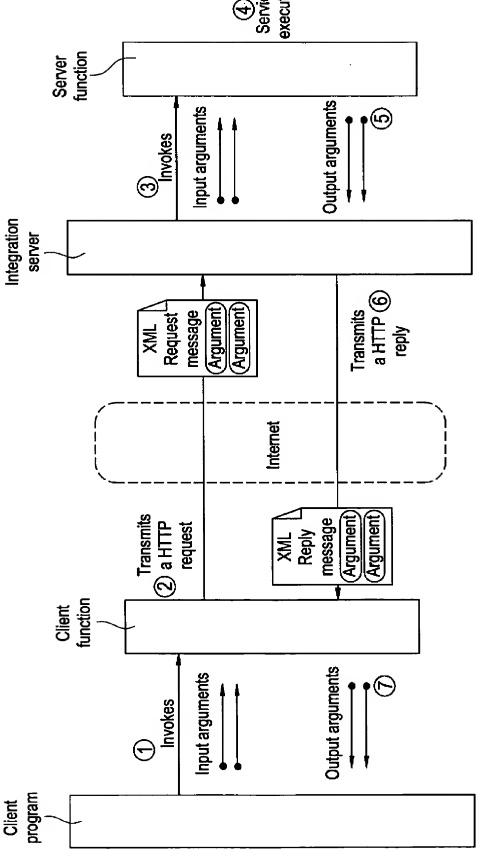
FIG. 6


Subsystem
Client program
1 Invokes
Input arguments
Output arguments
10
Stub function
2 Stub encodes message & sends to ORB
Encoded request message
Encoded reply message
ORB sends to stub & stub decodes 9
Object request broker (ORB)
3 Transmits message to server
IIOP Request message
Network
IIOP Reply message
Transmits message to client 8
Object request broker (ORB)
4 ORB sends to skeleton, which decodes
Encoded request message
Encoded reply message
Skeleton encodes & sends to ORB 7
Skeleton function
Server program
5 Invokes
Input arguments
Output arguments
6
Server function

R-820

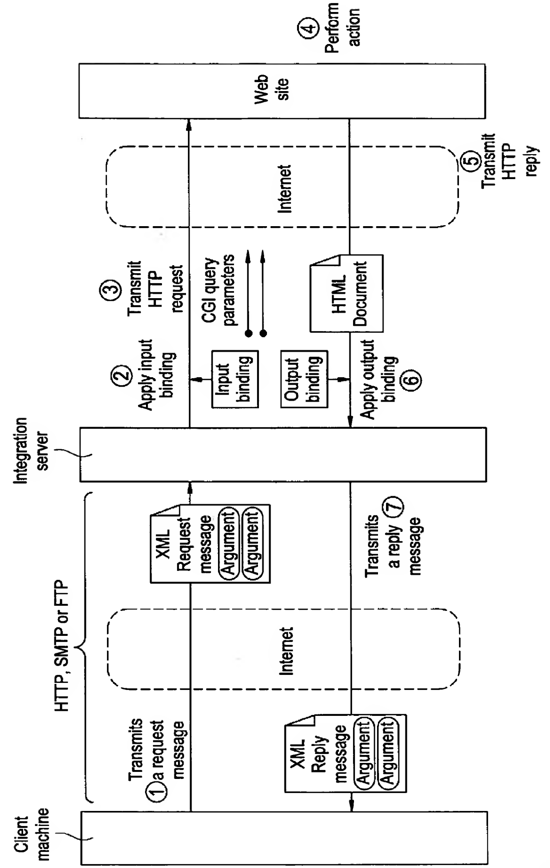
7/13

FIG. 7



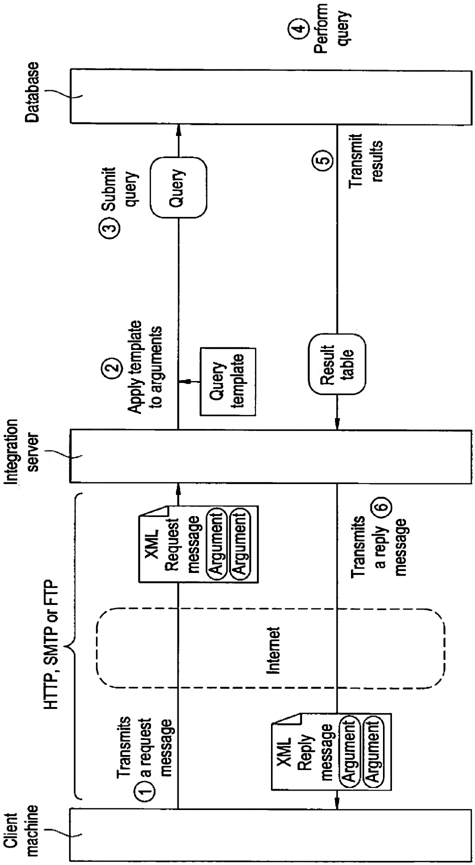
R-821

FIG. 8


Client machine
Integration server
Web site
HTTP, SMTP or FTP
Internet
Internet
1 Transmits a request message
XML Request message
Argument
Argument
2 Apply input binding
Input binding
Output binding
3 Transmit HTTP request
CGI query parameters
4 Perform action
5 Transmit HTTP reply
HTML Document
6 Apply output binding
7 Transmits a reply message
XML Reply message
Argument
Argument

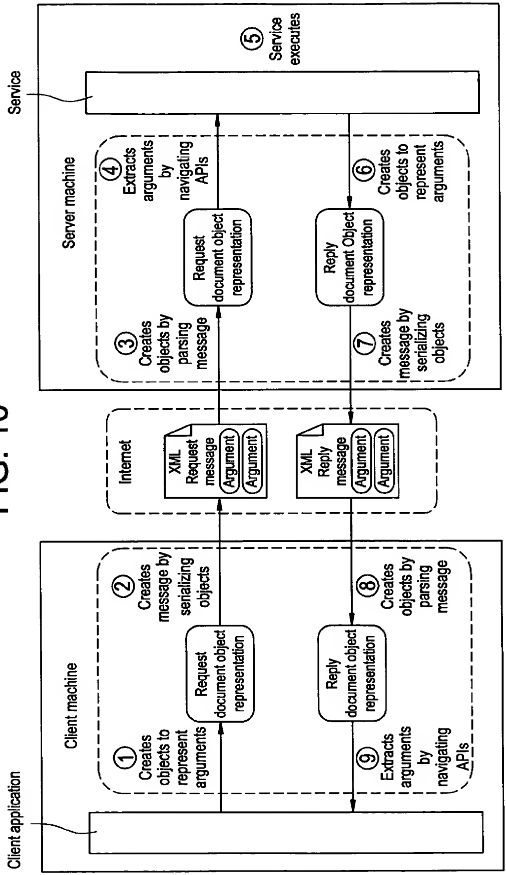
R-822

FIG. 9


Client machine
Integration server
Database
HTTP, SMTP or FTP
Internet
1 Transmits a request message
XML Request message
Argument
Argument
2 Apply template to arguments
Query template
3 Submit query
Query
4 Perform query
5 Transmit results
Result table
Transmits a reply message 6
XML Reply message
Argument
Argument

R-823

FIG. 10


Client application
Client machine
1 Creates objects to represent arguments
Request document object representation
2 Creates message by serializing objects
Reply document object representation
8 Creates objects by parsing message
9 Extracts arguments by navigating APIs
Internet
XML Request message
Argument
Argument
XML Reply message
Argument
Argument
Server machine
3 Creates objects by parsing message
Request document object representation
4 Extracts arguments by navigating APIs
7 Creates message by serializing objects
Reply document Object representation
6 Creates objects to represent arguments
Service
5 Service executes

R-824

FIG. 11

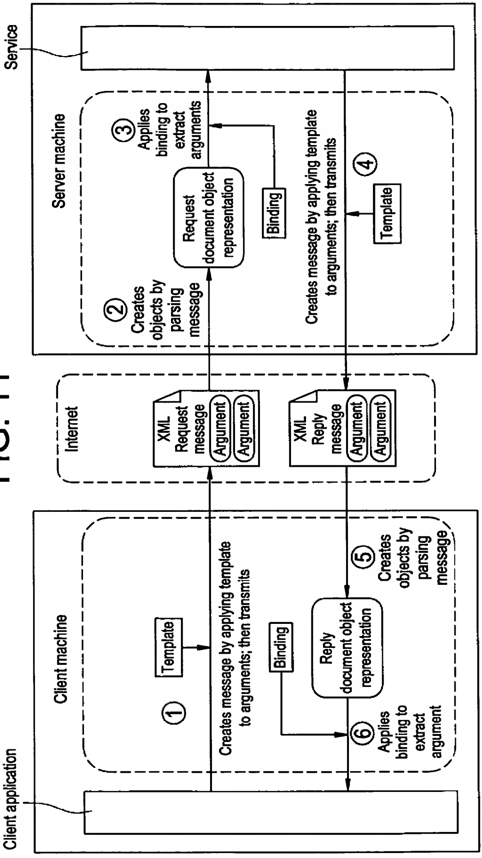
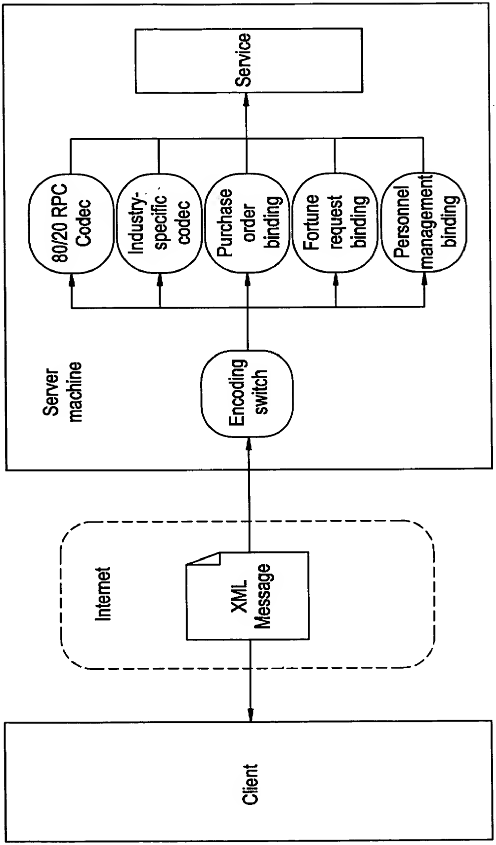
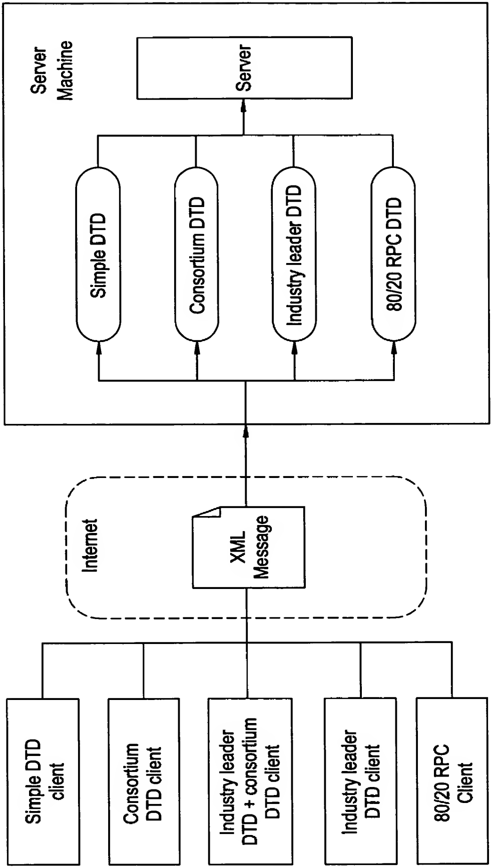

Service
Server machine
2 Creates objects by parsing message
Request document object representation
3 Applies binding to extract arguments
Binding
Creates message by applying template to arguments; then transmits
4
Template
Internet
XML Request message
Argument
Argument
XML Reply message
Argument
Argument
Client machine
Template
1
Creates message by applying template to arguments; then transmits
Binding
Reply document object representation
5 Creates objects by parsing message
6 Applies binding to extract argument
Client application

FIG. 12


Client
Internet
XML Message
Server machine
Encoding switch
80/20 RPC Codec
Industry-specific codec
Purchase order binding
Fortune request binding
Personnel management binding
Service

R-826

FIG. 13


Server Machine
Server
Simple DTD
Consortium DTD
Industry leader DTD
80/20 RPC DTD
Internet
XML Message
Simple DTD client
Consortium DTD client
Industry leader DTD + consortium DTD client
Industry leader DTD client
80/20 RPC Client

R-827

XML REMOTE PROCEDURE CALL (XML-RPC)

The present application is based on and claims priority from Application No. 60/079,100 filed Mar. 23, 1998 and Application No. 60/096,909 filed Aug. 17, 1998, the disclosures of which are incorporated by reference herein.

BACKGROUND OF THE INVENTION

Remote Procedure Call (RPC) is a mechanism by which an application residing on one machine requests the services of an application residing on another machine. A machine (as the term is used here and in the appended claims) may be a physical CPU or computer, a virtual CPU or virtual computer (otherwise known as a virtual machine or VM), or it may be a space of addressable memory in a system that isolates applications or processes so that they may not directly invoke one another. RPC requires one application to send one or more messages to another application in order to invoke a procedure of the other application. The recipient application may reply by sending one or more messages back to the requesting application. The term "RPC" identifies both the mechanism by which this occurs and the instance of such an exchange between applications.

Applications engaged in RPC conventionally do not message each other directly. Instead they rely on intervening software called "middleware" to translate between the wire protocol and the programming-language-specific functions or class methods of the application. This application uses the term "service" to generically refer to a function (including a procedure or a class method) or any action to be taken or performed, including access to web sites and databases. The term "wire protocol" refers to the transfer-protocol/message-encoding combination that the programs use as their common language for communication. The transfer-protocol is the mechanism by which the message transfers between the applications, and the message-encoding is the format in which the message is represented. The Object Management Group's IIOP (OMG Internet Inter-ORB Protocol) and the Open Software Foundation's DCE (OSF Distributed Computing Environment) are examples of wire protocols, and each defines both a transfer-protocol and a message-encoding. RPC has in the past been implemented using IIOP, DCE, or proprietary protocols. HTTP is an example of a transfer-protocol that is independent of message-encoding. The IIOP and DCE protocols have themselves been encoded into messages that tunnel through HTTP.

Each message-encoding is a particular format for representing a message. Any set of data may be represented in a particular format, not just RPC messages. This application uses the term 'data-package' to refer to each such set of data, and it uses the term 'encoding' to refer to the format in which a data-package is represented. An RPC message is a kind of data-package, and a message-encoding is the encoding of an RPC message. A tab-delimited file is an example of a simple encoding. The binary representation of a C programming language data structure is an example of a more complex encoding. A data-package may be thought of as a collection of individual data items. In a tab-delimited file, each row is a data item, as is each field in each row. In a C data structure, each field is a data item, as is each nested data structure and each field in each nested data structure. Notice that a data item may contain one or more other data items.

An 'argument' is a particular kind of data item. The term argument is used in the software industry to refer to a data item that is passed to a function or that is returned from a function. When the signature of a function is represented in the Java programming language, the arguments of the function consist of the comma-delimited terms that appear in parentheses and the term that represents the return value of the function. Consider the following function signature:

EmployeeID addEmployee(Name n, Address a, JobType t)

This function has four arguments whose classes are EmployeeID, Name, Address, and JobType. The Address class might be defined as follows:

```
class Address {
    String street;
    String city;
    String state;
    String zip;
}
```

In other contexts one might interpret the street, city, state, and zip data items to each be an argument of the function, but this application defines 'argument' such that these data items are not arguments. The arguments of a function are those data items that the signature of the function identifies, where for purposes of this definition the definitions of the argument types are not considered to be part of the signature.

A function is only one kind of service, and XML RPC messages can exist in the absence of functions. Hence, it is necessary to define the term 'argument' in terms of the message encoding. Therefore, given a data item X contained in message M1 expressed in encoding E, X is an argument when it satisfies both of the following requirements: (1) there exists a message M2 also expressed in E such that M2 is constructed by adding a data item Y to M1 where neither X contains Y nor Y contains X; and (2) X is not contained in any data item having property (1). It is possible to define an encoding that hard-codes arguments relative to another encoding so that the defined encoding expresses a subset of the messages of the other encoding. Consequently, a message cannot be examined for arguments without interpreting the message with respect to a particular encoding.

Each argument is an input argument, an output argument, or both an input and an output argument. The return value is always an output argument, but function signatures usually do not indicate which if the remaining arguments are input arguments or output arguments. The input arguments and output arguments must be established by convention. The input arguments are those arguments that the software that invokes the function uses to pass data items to the function. The output arguments are those arguments that the function uses to pass data items to the software that invoked the function. It is possible for one argument to be used for both purposes. In this case, the argument appears in both the RPC request message and the RPC reply message, although the values of the occurrences may differ. The OMG CORBA specification provides further treatment of the distinction between input and output arguments in the context of RPC.

In addition to containing data items, a data-package may contain labels. A label is an identifier that provides information about one or more data items. A label that provides information about a data item is said to be a label for the data item. No data item is a label, since the encoding assumes responsibility for providing the association between the data item and the label.

This application defines two kinds of labels: type labels and semantic labels. A type label identifies the data type

each associated data item. Data types include the primitive data types that programming languages define. The primitive data types include integers, floats, longs, strings, and booleans. According to this definition, data types also include primitive data structures, such as arrays, records, and vectors (e.g. the Java Vector class). Data types provide the information that is necessary to represent the data item in memory so that a programming language may use the data item. However, there is no requirement that any two programming languages or any two implementations of RPC represent the same data types in the same way.

A semantic label is an identifier that names an associated data item according to the data item's role within the message. The purpose is to allow software that consumes the message to locate data items by role. The semantic label of an argument is analogous to the name that a method signature gives to the argument. Again consider the following signature:

EmployeeID addEmployee(Name n, Address a, JobType t)

This function has arguments named 'n', 'a', and 't'. It also has an unnamed argument whose type is EmployeeID. The names may be used as semantic labels for their associated arguments. The semantic label of a field found in a class, record, or structure is analogous to the name given to that field. Consider the previously given definition of the Address class. It has four data items. Each has a data type of string, and their names are 'street', 'city', 'state', and 'zip'. These names may be used as the semantic labels for the associated data items. However, unlike the names of a programming language structure, multiple data items may have the same semantic labels. For example, two data items may each have the 'street' label to indicate that they each represent one line of the street address. On the other hand, an array or a vector may contain multiple data items such that a semantic label is associated with the entire array or vector but not with each data item.

This application classifies encodings as either self-describing or non-self-describing. A self-describing encoding associates a semantic label with each argument and includes both the label and the association within each data-package. A completely self-describing encoding associates a semantic label with every data item except possibly for the data item constituents of arrays and vectors. A semantic label allows a program to identify a data item without requiring the program to know in advance the location of the data item within the data-package and without requiring the program to know the exact structure of the data-package. One example of a data-package that uses a self-describing encoding is a tab-delimited file that begins with a row consisting of the names of each field in a row. Another self-describing encoding is the binary representation of a hash table, which assigns a name (or key) to each data item in the table. Non-self-describing encodings do not contain semantic labels for their arguments. A program that reads a data-package expressed in a non-self-describing encoding must assign semantics to each argument by applying information found in the application and not in the data-package. For example, a program that reads a tab-delimited file to load records into a database must know in advance how to correlate the fields in the file to the columns of the database table.

This application also classifies encodings as either specific or generic. A specific encoding is an encoding that is designed to represent an application-specific or industry-specific data structure. A generic encoding is an encoding that is designed to represent any or nearly any data structure, regardless of application or industry. For example, the ASCII form of a database report is a specific encoding since it can express the data found in a particular database but not the data found in any database. Packet headers in communications protocols also use specific encodings. An implementation of the C programming language uses a generic encoding to represent in-memory data structures; this is a generic encoding because one encoding scheme is capable of representing any C data structure.

Finally, this application distinguishes between binary encodings and text-based encodings. A text-based encoding is a human-readable and human-writable encoding. Data-packages expressed in a text-based encoding can be loaded into a text editor and can be read and modified by humans using the text editor. A human can also create a data-package expressed in a text-based encoding by writing the message completely from scratch using the text editor. Text editors include vi, emacs, MS-DOS EDIT.EXE and Microsoft Notepad. Binary encodings are those encodings that do not satisfy these human-accessibility criteria. Tab-delimited files are examples of data expressed in a text-based encoding, while the in-memory representation of C data structures is an example of a binary encoding.

The IIOP and DCE wire protocols use non-self-describing generic binary encodings. Of all the example encodings discussed in this application, the IIOP and DCE encodings are most closely analogous to the in-memory binary representations of C or C++ data structures. However, the analogy is not perfect because the encodings have been designed to support the data structures found in a variety of languages, and because they have been designed to support the translation of data between a variety of platforms.

Although IIOP and DCE are non-self-describing, it is possible to define services in IIOP and DCE that accept self-describing parameters and that invoke other services using the identified parameters. Microsoft's COM (Component Object Model) accomplishes this on top of DCE through the set of services found in its Dispatch interface. By invoking the services of IDispatch, an application can emulate the functionality that would be available through a wire protocol that is natively self-describing. However, DCE remains non-self-describing, since the services of IDispatch are expressed in the same non-self-describing encoding in which all other service invocations of the protocol are expressed. IDispatch effectively defines a high level protocol that is layered on top of DCE. More work is required to develop software that invokes self-describing services through IDispatch than is required to develop software that invokes the native non-self-describing services of DCE. Additional effort is required because to invoke a single self-describing service the developer must write software that conforms to the IDispatch protocol. Invoking a single non-self-describing service only requires the developer to invoke a single function in the code that DCE IDL compilers automatically generate for the developer.

IIOP defines a data-type called 'ANY'. A data item expressed in this type is a container for other data items. The ANY data type associates a type label with each data item that it contains, but it does not associate a semantic label with any of these data items. It also does not associate a type label with the instance of the ANY data item itself, unless the instance happens to be contained within another ANY data item. Consequently, an implementation of IIOP that receives a message containing an instance of the ANY data type cannot determine that the instance is of type ANY simply by examining the message. The recipient of the IIOP message

must know in advance that the argument is of type ANY. In fact, the recipient must know the types of all the arguments in order to extract the arguments from the message. This aspect of IIOP makes programs that use IIOP strongly sensitive to changes in the messages. Adding, removing, or changing argument types in the programs that generate the IIOP messages necessitates making analogous changes in all the programs that receive the messages. This would not be necessary in programs that relied on data type labels that the message itself provides.

IIOP and DCE are also stateful, bi-directional, and complex. These protocols are stateful because the client and server must maintain state information to successfully transfer messages between them. The protocols are bi-directional because the server must sometimes asynchronously send messages to the client. Finally, IIOP and DCE are complex because they are designed to provide the union of the features found in a wide variety of programming languages. This latter property of IIOP and DCE allows any two programs to communicate with service invocations that support the rich variety of invocation mechanisms that are common to the programming languages in which the two programs are written.

Traditional middleware implementations are based on the IIOP and DCE protocols and consequently suffer from the following drawbacks:

- Because the encodings are generic, translation software is required to allow RPC messaging between a program that uses a specific encoding and a program that uses a generic encoding. Translation software is also required between programs that use different specific encodings. If IIOP or DCE is to be the wire protocol, the translation software must be installed between each program that uses a specific encoding and the communications channel that connects the programs. Furthermore, generic encodings are necessarily more removed from the data structures that are common to a given industry, which prevents most industry experts from examining messages, since expertise with the generic encoding would be required in addition to expertise with the industry data structures.
- Because the encodings are binary, special software is required to read, modify, or write the RPC messages. If the message is expressed in a non-self-describing encoding, the special software must be application-specific or industry-specific software. These factors limit the accessibility of messages expressed in the IIOP and DCE encodings. since the translation software must exist, and since one must have translation software on hand to access the messages.
- Because these encodings are non-self-describing, programs can't extract and selectively process the data items of an arbitrary message without first being specifically configured to recognize the particular kind of message. Programs that do support configured access to non-self-describing messages generally require separate configuration information for each kind of non-self-describing message. As the message-generating programs evolve the messages and add new message types, the configuration information of the other programs must be updated. To further complicate matters, each program may require different configuration details or separate configuration entry procedures. Finally, if a message-generating program changes the format of a non-self-describing message, the message-consuming programs must also change their knowledge of that message, since otherwise the these programs would not be able to extract and identify the data items found in the message.
- Because the IIOP and DCE protocols are stateful, the protocols are not well-suited for messaging using HTTP, since HTTP is a stateless protocol. HTTP provides only a limited mechanism for maintaining client state. Under this mechanism, the client maintains the state information, which means that client-provided state information cannot be trusted to belong to the same client over time, unless the client connects through an authenticated connection. One must therefore inventively design a stateful protocol on top of HTTP in order to tunnel IIOP or DCE through HTTP. It is desirable to use HTTP as the transfer protocol because of the ubiquity with which it is installed on operating systems, and because it is typically the only protocol that network firewalls allow to pass.
- Because these protocols are bi-directional, the protocols are not well-suited for messaging using HTTP, since HTTP is unidirectional. For example, an IIOP client may register a callback function with an IIOP server. When it comes time for the server to invoke the callback function, the server cannot issue an HTTP request to the client unless the client is also configured to be an HTTP server. Administrative and security requirements normally make it undesirable to configure a client as an HTTP server. Therefore, one must inventively design a bi-directional protocol on top of HTTP in order to support IIOP and DCE bi-directional behavior.
- Because the encodings are intended to bridge applications written in almost any two programming languages on almost any two platforms, the encodings are proportionately complex. IIOP and DCE take the superset approach to bridging applications by ensuring that the encoding can represent nearly every kind of service invocation that programming languages are capable of expressing. To create software that uses IIOP or DCE, a software developer must define the signatures of these services. The superset approach complicates the definitions and minimizes the pool of skilled experts that can develop for IIOP and DCE. This approach also complicates the tools that manipulate messages expressed in these encodings.
- Because of the complexity of the IIOP and DCE protocols, implementations from two different vendors are generally incompatible. To get two programs to communicate over a network, the computers running the programs must conventionally run middleware software from the same vendor. Vendors have implemented gateway software to bridge between different implementations of these protocols. The CORBA 2.0 specification has mitigated this problem to a large degree, but it has not eliminated it completely. Thus, integrating programs residing in different organizations often requires either both organizations to agree on the installation or one company to buy additional software to provide a gateway between the existing middleware installations.

Therefore, there is a need for improved RPC techniques that do not suffer from the above drawbacks. The techniques would provide the following benefits:

- It would be beneficial to have an RPC mechanism that allows messages to be expressed in application-specific or industry-specific encodings. This would allow programs to communicate without requiring mes-

expressed in specific encodings to be translated into messages expressed in generic encodings. Provided that the messages were also text-based, this would also allow industry experts to create and examine messages without requiring that they have expertise in a generic encoding. Text-based messages in specific encodings would also be easier to read than text-based messages in generic encodings, since the generality of generic encodings makes generically-encoded messages more verbose.

It would be beneficial to have an RPC mechanism that uses text-based messages, since special software would not be needed to read, write, or modify the messages. Text editors would suffice. People who have expertise in industry data structures would have less reliance on the expertise of others to translate messages into a form that they can understand.

It would be beneficial to have an RPC mechanism that is based on a self-describing encoding such that all messages are at some level expressed in the self-describing encoding. This approach would simplify the programs that access the data items of the messages, since the programs would no longer require configuration information. Furthermore, because the programs that generate and consume the messages could access data items by label, the programs would be less sensitive to changes in the message types.

It would be beneficial to have an RPC mechanism that operated statelessly over HTTP so that less complex software is required to implement RPC over HTTP.

It would be beneficial to have an RPC mechanism that did not support bi-directional communications so that less complex client software is required to implement RPC over HTTP. A unidirectional RPC would not support asynchronous callbacks, and an application that is based on the RPC would have to either select a synchronous solution or install an HTTP server at both ends of the communication channel.

It would be beneficial to have a generic encoding that precludes some of the less widely used features of IIOP and DCE so that the encoding is simpler than IIOP and DCE. It would be beneficial for this generic encoding to contain only those features that are necessary for most business-to-business electronic commerce over the Internet, thus idealizing the encoding for this domain of applications. The approach maximizes the number of software developers that are qualified to produce solutions with the encoding and minimizes the time required to produce each solution.

It would be beneficial to have an RPC framework whose simplicity allows two programs to engage in RPC communications over a network without requiring software from a single vendor to be installed on both of the computers that are running the programs. Software from different vendors should be compatible by virtue of using the nearly ubiquitous HTTP protocol stack to transmit messages whose encodings use a well-defined and well-accepted text-based syntax.

SUMMARY OF THE INVENTION

It is therefore an object of the present invention to provide an improved remote procedure call mechanism, which is both versatile yet simple to implement. The invention is directed to both the method of remote procedure call as well as a system for implementing the method.

It is a further object of the invention to provide a generic message encoding which is less complex.

These and other objects are achieved, according to a first aspect of the present invention, by a method and system for invoking a service at a first machine from a second machine, in which the message encoding is self-describing. In a preferred embodiment, the message encoding is XML, although the invention is not to be so limited. The method according to the invention employs type and/or semantic labeling of arguments and/or data items within arguments. The invention extends to the client sending a request, or a server responding to a request. In one implementation, all arguments and all contained data items are both type and semantically labeled, although labeling, particularly semantic labeling, may be dispensed with for data items contained in an argument, with these data items instead assuming the attribute of the containing argument.

The invention is particularly applicable to remote procedure call (RPC), although it is applicable as well to other types of service invocation. The service invocation request and service invocation reply need not use the same message encoding, or even the same transfer protocol. The method of the invention, and particularly its message encoding, may be used with any of a number of transfer protocols, e.g., HTTP, FTP and SMTP, with its widest use at present being in conjunction with HTTP. In this context, the service to be invoked may be designated in the HTTP header, or in the URL. Alternatively, the service to be invoked may be identified within the message encoding.

According to a further aspect of the invention, a message encoding is provided which is simple yet functional. It is a markup language-based, self-describing message encoding using a set of element type labels which is small, e.g., six in one preferred embodiment, designed to maximize functionality with a minimum of complexity. In addition to labeling each data item with one of the message encoding labels, additional descriptive information for each data item can be provided. The novel encoding is independent of the syntax used to express the encoding, but in the illustrative example the message encoding is XML. In such a case, the additional information for each data item may be provided in the form of XML attributes.

The preferred embodiment uses element types having the following attributes: a VALUE element representing a data item that is a lexical unit, and it has an optional TYPE attribute. The value of the optional TYPE attribute that names the lexical type. VALUE elements that lack the TYPE attribute are assumed to be strings; a RECORD element containing a set of one or more named elements of any type, with each contained element having a NAME attribute, and the value of each NAME attribute being unique among the values of the NAME attributes of all children of the containing RECORD element; a LIST element containing a set of elements which do not have names; or if they do have names, the names are ignored; an OBJECT element referencing any other element by a unique object ID, and serving as a placeholder for the element it references; and NULL element representing a VALUE, a RECORD, a LIST, or an OBJECT for which no data is provided: and an ARRAY element representing a single or multi-dimensional array of elements, and having an optional DIMENSION attribute and an optional TYPE attribute, with the DIMENSION attribute indicating the dimensionality or depth of the array, and the TYPE attribute indicates the type of data found in the array.

Of particular note is the use of an ID attribute and OBJECT element, by which a single data item can be contained in multiple data items, with the ID attribute and

the OBJECT element allowing RPC messages to express arbitrary directed graphs, including cyclic graphs.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention will be more clearly understood from the following description in conjunction with the accompanying drawings, in which:

FIG. **1** is a diagram depicting the overall architecture of XML RPC;

FIG. **2** is a diagram depicting the role of the an Integration Server in the XML RPC architecture;

FIG. **3** is a diagram depicting a scenario in which an integration server is both a client and a server of XML RPC;

FIG. **4** is a diagram illustrating the use of XML RPC over HTTP, SMTP, and FTP;

FIG. **5** is a diagram depicting the isolation of the wire protocol layer from the service layer at the server;

FIG. **6** is a diagram illustrating the OMG CORBA mechanism for implementing RPC;

FIG. **7** is a diagram illustrating the use of XML RPC to invoke functions;

FIG. **8** is a diagram illustrating the use of XML RPC to invoke a web site service;

FIG. **9** is a diagram illustrating the use of XML RPC to invoke a database operation;

FIG. **10** is a diagram illustrating the use of codecs to implement XML RPC;

FIG. **11** is a diagram illustrating the use of templates and bindings to implement XML RPC;

FIG. **12** is a diagram depicting the architecture of a program that is capable of supporting XML RPC messages expressed in a variety of encodings; and

FIG. **13** is a diagram depicting the benefits of the architecture shown in FIG. **12**.

DETAILED DESCRIPTION OF THE INVENTION

Several key aspects of the invention relate to the use of eXtensible Markup Language (XML). XML is a simplification of the ISO Standard Generalized Markup Language (SGML). SGML is a standard document formatting language that enables a publisher to create a single document source that can be viewed, displayed, or printed in a variety of ways. SGML is a large, complex formatting language that is not well suited to the Web. The World Wide Web Consortium (W3C) created XML to make SGML-like functionality available to the broad audience of Internet users. XML is well-understood in the industry, and one can acquire a detailed understanding of it from any number of public sources.

Like SGML, XML defines a class of markup languages rather than a single markup language. Unlike HTML, which defines a set of markup tags such as <P> and <TABLE>, XML defines the syntax of a markup language without defining the tags that constitute any language. For example, HTML does not conform to the XML syntax, but it does closely conform to the SGML syntax. HTML may therefore be thought of as one of many SGML languages. Likewise, one may define many different XML languages, each with its own set of tags. A data-package expressed in XML is known as a document, and an XML language is known as a document type. A document type definition (DTD) defines the tags that are valid in a particular document type along with the valid relations among those tags. SGML uses the same terminology, so HTML is a document type of SGML.

The originally intended purpose of XML is the same as the originally intended purpose of SGML, except that XML was designed to be much simpler. XML allows one to label the information in a document so that one may retrieve the information contained in the document according to its label. Labeling schemes may be chosen according to the requirements of the programs that process the documents. Consider a document expressed in the HTML document type:

The catalog contains these products:<P>
Model <B>233A</B> costs <B>$40.00</B>.<P>
Model <B>124B</B> costs <B>$125.00</B>.

Compare this document to the following, which is expressed in an XML document type that uses special tags not defined in HTML:

```
<CATALOG>
    The catalog contains these products:
    <P>Model <MODEL>233A</MODEL> costs
        <PRICE>$40.00</PRICE>.</P>
    <P>Model <MODEL>124B</MODEL> costs
        <PRICE>$125.00</PRICE>.</P>
</CATALOG>
```

The tags of the HTML document do not clearly label the information they enclose. The <B> tags enclose both the model number and the price, and no distinction is made between products and paragraphs. The XML document uses tags with names that describe the information the tags enclose. One may retrieve the model numbers by retrieving the contents of all MODEL tags, and one may retrieve the prices by retrieving the contents of all PRICE tags. XML allows one to define document types that clearly label the contents of a document.

By descriptively labeling the contents of a document, one may store the document and subsequently put the document to many different purposes. Suppose an XML document has been made available on the Internet. One program may download the document, extract the prices, and perform a calculation on the prices. Another program may look for the MODEL and PRICE tags and render the contents of the document on the screen such that the models and prices are shown in bold. Publishers make heavy use of this feature by leaving the document unchanged while changing the style in which the document is printed. Still another program may search the document for a particular model to return its price. The document can be put to many uses and is thus a significant improvement over word processing documents, which do not differentiate text. XML is therefore well-suited for the publication industry.

XML document types provide additional advantages over HTML. The syntax of HTML is more lax than that of XML, and there are many expressions in HTML whose parses are ambiguous. The above two documents illustrate one ambiguity. The HTML document shown does not close its <P> tags. The following interpretation is therefore possible:

The catalog contains these products:<P>
Model <B>233A</B> costs <B>$40.00</B>.<P>
Model <B>124B</B> costs <B>$125.00</B>.
</P></P>

In this interpretation, one <P> element contains the other as a child. Ambiguities such as these limit one's ability to use the document with many tools, since each tool may interpret

the document differently. XML eliminates these ambiguities by enforcing strict syntax and thus improves software interoperability.

XML documents consist primarily of three syntactic features: elements, attributes, and character data. An element consists of a start tag, an end tag, and all data between the start tag and end tag. Each element has an element type name, which is the name that identifies the element within a tag. In the above XML document, the element type names are CATALOG, P, MODEL, and PRICE. An element start tag may optionally contain a set of attributes. Each attribute consists of an attribute name and an attribute value. Attributes also exist in HTML. Finally, an element may contain character data between its tags. These are the characters that do not appear in either within any tag. XML requires that every document have a root element which contains all character data and all other elements of the document.

Although XML was designed to represent documentation, one may also use it to represent data. Consider the following XML document:

```
<CATALOG>
    <PRODUCT><MODEL>233A</MODEL>
        <PRICE>$40.00</PRICE></PRODUCT>
    <PRODUCT><MODEL>124B</MODEL>
        <PRICE>$125.00</PRICE></PRODUCT>
</CATALOG>
```

If the tags were removed, the document would be difficult for humans to read:

233A$40.00124B$125.00

The difference is that the document representing data does not use XML mixed content. Mixed content occurs when elements and character data occur together as immediate children of the same parent element. Although mixed count is useful in documentation, it is not useful in documents that represent data, since raw data is intended for program consumption rather than for consumption by humans.

Certain properties of XML are particularly important to the present invention. These properties are also common to other markup languages. As used here and in the appended claims, a markup language is a syntax for expressing encodings such that the syntax has all of the following properties:

(1) The syntax partitions its expressions into sections comprising data and sections comprising metadata (herein referred to as 'markup').

(2) The syntax defines a structure (herein referred to as an 'element') that unambiguously collects data and markup sections into a group of sections by using markup to delimit the group from surrounding sections.

(3) The syntax defines markup that allows an expression to associate a label (herein referred to as "element type name") with any element.

(4) The syntax allows elements to nest within elements.

Web Interface Definition Language (WIDL) is an XML document type for program consumption developed by the assignee of the present application. This document type serves a number of purposes. One purpose is to provide an Interface Definition Language (IDL) expressed in XML. IDLs are well-known to the distributed computing community. They are languages for defining the signatures of program functions in a way that is independent of programming language. WIDL refers to these functions as "services." Another purpose is to implement the service described by the signature. An early version of WIDL interpreted the service as a means for programmatically interacting with a web site. When a programming language called the service, the input arguments to the service were passed as CGI query parameters to a target web site. The web site returned an HTML document, and the service implementation described how to extract data from the HTML document so that the data constituted the output arguments of the service. A program could therefore invoke the service to perform an action on a web site without requiring the program to have any knowledge of the web site or of the network protocols necessary to engage the web site.

The present assignee has developed XML RPC and a Business-to-Business Integration Server (B2B), while also enhancing WIDL to better support interaction with XML documents. XML RPC is an RPC mechanism that uses XML documents as the request and reply messages. A client machine sends a message to a server machine to ask the server machine to invoke a service. The server machine invokes the service and sends a reply message to the client. FIG. **1** illustrates the mechanism. In step **1**, a client machine generates and transmits a request message to a server machine. In step **2**, the server machine interprets the message as a request to invoke a service and invokes the service. In step **3**, the server machine sends a reply message to the client machine. Step **3** is optional for the case where the client machine does not require a reply message.

FIG. **2** illustrates the architecture that B2B uses to implement XML RPC. In this case the server machine runs as an integration server, of which the B2B server of the present assignee is an example. The integration server interprets a request message as a request to invoke a service and invokes the service on behalf of the request message. When the service completes and provides invocation results, the integration server receives the invocation results and sends a reply message to the client machine on behalf of the service. In step **1**, the client machine sends a request message to the integration server. In step **2**, the integration server translates the message into a service request, formatting the service request as required by the service. In step **3**, the service performs the request. In step **4**, the service returns the results of the request to the integration server. In step **5**, the integration server translates the results into a reply message, formatting the message as required by the client machine, and sends the reply message to the client machine.

FIG. **2** portrays the integration server as the system with which the client machine communicates. An integration server may also reside on a client machine and implement the XML RPC client mechanism. B2B provides this functionality. FIG. **3** illustrates one such scenario. In this case, Integration Server B and Service Y together constitute the service that Integration Server A invokes. The diagram labels this composite service Service X. The machine on which Integration Server A resides assumes the role of the client machine. The machine on which Integration Server B resides assumes the role of the server machine. This diagram therefore depicts two integration servers engaged in XML RPC. Integration Server A passes the XML RPC request on to Integration Server B, either by passing the request message unchanged or by translating the request message into a new message having a document type that Integration Server B accepts. Integration Server B acts on the message and then sends a reply message back to Integration Server A. Integration Server A then passes the reply back to the client machine. possibly performing a translation on the message before transmitting it.

The B2B integration server provides two mechanisms for performing XML RPC. One mechanism uses programming modules called "codecs." Each codec recognizes a particular XML document type. An XML document type is also referred to as an XML-based encoding. The codec is responsible for generating an XML document from a set of arguments and for extracting a set of arguments from an XML document. Codecs are most useful with XML document types that this application later refers to as "generic" encodings. The other mechanism uses templates and WIDL. B2B uses a template to generate an XML document from a set of arguments, and it uses a binding to extract a set of arguments from an XML document. This latter mechanism is most useful with XML document types that this application later refers to as "specific" encodings. More information about codecs, templates and bindings is available later in the application.

WIDL is described in detail in Charles Allen, *WIDL: Application Integration with XML*, World Wide Web Journal, Vol. 2, Issue 4, 1997, pp. 229–248; and Mark Wales, WIDL: Interface Definition for the Web, IEEE Internet Computing, January–February 1999, Pages 55–59. A complete specification of WIDL is also found in B2B Developer User Guide, webM-DEV-U6-990311, available from the present assignee. All of these publications are incorporated by reference herein. A comprehensive description of XML RPC can be found in *The XML Handbook*, chapters 8 ("Supply Chain Integration") and 38 ("WIDL and XML RPC"), published by Prentice Hall in 1998, both of which are also incorporated by reference herein.

XML RPC

A first improvement according to the present invention is the implementation of Remote Procedure Call (RPC) using an XML-based message-encoding. XML was designed to represent documents for use in the publishing industries (including web-based publishing) and was not designed for use as the syntax of an RPC message-encoding. The present inventors have found that use of XML-based message-encodings provides many benefits, including the following:

- One may design document types that are specific to a particular application or industry. Even though the encodings are not generic encodings, programs may still generically extract the data items that the messages contain, since the messages are expressed in the universally defined syntax of XML.
- One may design document types that are also text-based encodings. Although every XML-based message can be loaded by any XML text editor, it is ultimately the responsibility of the designer of an encoding to ensure that the values of data items are human-readable, since it is possible to cryptically encode data in human-readable text characters. However, because the XML syntax is inherently human-readable, XML naturally biases encoding designers to ensure that the values of all data items are also human-readable.
- All XML-based encodings are inherently self-describing, since XML organizes data items into named elements and into named element attributes. However, the descriptiveness of the labels varies by encoding, since it is possible for encoding designers to choose poorly differentiated names for the data items. Encodings having well-differentiated names can minimize or even eliminate the need for programs to be configured to recognize the encodings.

An RPC protocol must include a transfer protocol. This application defines a transfer protocol as a mechanism for transferring messages between communicating programs. XML RPC may use any of a number of known transfer protocols, including network-specific protocols such as HTTP, SMTP, or FTP. FIG. **4** illustrates this process for the case where the transfer protocol is one of HTTP, SMTP, or FTP and the messages are transmitted across the Internet. According to the XML RPC procedure, and as depicted in step **1** of the figure, a client uses the transfer protocol to send an XML-based message to a server. This message is known as the request message. When HTTP is the transfer protocol, it is natural to accomplish this via HTTP's POST method. As shown in step **2**, the server interprets the request message as a request to perform a particular service and performs the service. The server may perform this service using information found in the request message. When the service completes, the server generates a reply message. The reply message may contain information describing the results of the service. In step **3** the server then transmits the reply message to the client that initiated the request.

Implementing XML RPC over SMTP and FTP is straightforward provided that the developer synchronizes messages sent with messages received. Consider SMTP. SMTP is the Internet mail protocol, and using XML RPC over SMTP amounts to sending XML documents to servers via email and having servers send XML documents back to the client, also via email. Servers need only send a response to a request message if the request message merits a response, such as in the case where the service being invoked returns output arguments. SMTP does not guarantee the order in which messages will be delivered to a destination. Should the order of occurrence of invocation requests be important, the client will be responsible for identifying the order of invocation, and the server will be responsible for invoking the services in this order. The client must also associate reply messages received with request messages sent so that it can return the appropriate output arguments from the service that the client program invoked. Many solutions exist to this synchronization problem, as it is a common problem in network protocols.

One way to implement the RPC mechanism is to isolate the wire protocol layer from the service layer. One may isolate these layers on the client machine, on the server machine, or on both machines. FIG. **5** depicts an implementation in which the layers are isolated on the server machine. This application uses the term "integration server" to identify the layer that sits between the client machine and the service. In step **1**, the client machine generates a request message and sends it to the integration server. The request message contains a set of input arguments. In step **2**, the integration server extracts the arguments from the request message and passes the arguments to a service, invoking the service in the process. As shown in step **3**, the service then performs an action. The action may be based on the input arguments that the integration server provided to it. In step **4**, the service completes and returns output arguments to the integration server. The integration server generates a reply message that contains the output arguments and transmits the reply message to the client machine, as shown in step **5**.

Isolating the wire protocol layer from the service layer on the server machine provides quite a bit of flexibility. It allows an integration server to access many different kinds of services. First, consider the conventional RPC mechanism, for which we take OMG CORBA as an example. FIG. **6** portrays the mechanism by which OMG CORBA performs RPC. Here a client program and a server program are messaging via RPC. Each program is running on a machine that has an Object Request Broker (ORB) installed.

ORB is responsible for sending and receiving IIOP messages and for directing messages to the appropriate software component. CORBA defines two software components for this purpose: stubs and skeletons. A stub is a component that translates between a function signature and a message encoding on the client side, and a skeleton is a component that translates between a function signature and a message encoding on the server side.

FIG. 6 illustrates the steps of the conventional RPC mechanism. In step 1, a subsystem within the client program calls a function implemented by the stub, possibly passing input arguments to the stub. In step 2, the stub translates the input arguments into an IIOP encoding and hands the encoded arguments to the client ORB. In step 3, the client ORB sends the request message over a network using the IIOP protocol, and the server ORB receives the message. In step 4, the server ORB passes the encoded arguments to the appropriate skeleton, and the skeleton decodes the arguments. In step 5, the skeleton calls an appropriate server function, passing the arguments to the function as inputs. At this point the function executes. In step 6, the function returns output arguments to the skeleton. In step 7, the skeleton translates the output arguments into an IIOP encoding and then passes the encoded arguments to the server ORB. In step 8, the ORB sends a reply message over the network using the IIOP protocol, and the client ORB receives the message. In step 9, the client ORB passes the encoded output arguments to the stub, which decodes the arguments. Finally, in step 10 the stub function returns the output arguments to the subsystem that invoked the function.

The XML RPC architecture can emulate the conventional RPC mechanism. FIG. 7 illustrates the approach. The approach does not require a client-side ORB due to the simplicity of the protocol. The client need only use the HTTP stack and an XML parser, both of which are widely available and usually free. A client wishing to use a generic document type might utilize a utility component for this purpose, such as the codec module described below. Clients using custom document types would not need such a module. The XML RPC architecture also does not require a server skeleton. Type labels and semantic labels in the XML messages may provide the server with all of the information that the server needs to decode the message. Some XML message encodings may not provide all of the necessary information. Provided that the encodings semantically label the data items they express, it is possible to configure the server to handle these encodings using declarative mechanisms such as WIDL bindings.

The steps are a subset of the steps that CORBA IIOP requires. In step 1, the client program invokes a client function, possibly passing input arguments to the function. This step is not strictly necessary, since the simplicity of the protocol would allow the client program to itself perform the task. However, continuing the comparison with the conventional RPC mechanism, in step 2 the client function generates an XML request message from these arguments and transmits the message to the integration server. The diagram illustrates the use of HTTP to perform the transmission, but any transfer protocol will suffice. In step 3, the integration server automatically decodes the message and invokes a server function, passing the arguments as inputs to the function. As shown in step 4, when the server function completes, the integration server receives the output arguments. In step 5, the integration server automatically generates an XML reply message from the output arguments and sends this message to the client. In step 6, the client extracts the output arguments from the message and returns them to the client program that invoked the client function.

FIG. 8 illustrates the flexibility that an XML RPC solution allows. The previous example illustrating using XML RPC to invoke a server function. This example illustrates a case where the mechanism is invoking a service that is not a function. Here a web site is behaving as a service. One may view a web site as a service amenable to automated access even though web sites were intended for browser access. In a browser one may fill out a form, press a button to submit the form parameters to a server, and then get an HTML page back. The values entered into the form may be thought of as input parameters, the button may be thought of as the means by which a service is identified, and the HTML page may be thought of as a collection of output parameters. It is therefore possible for software to wrap a web site so that it looks like a function call. WIDL allows one to accomplish this. Given that a web site may be viewed as a service, given that XML RPC is a mechanism by which one may invoke a remote service, and given the self-describing properties available to an XML-based encoding, one may use XML RPC to invoke web services.

In step 1 of FIG. 8, a client machine transmits an XML request message to an integration server, where the request message containing a set of input arguments. The integration server extracts the input arguments from the message via one of the mechanisms specified later in this specification. In step 2, the integration server applies a WIDL input binding to these arguments to prepare an HTTP request. The input binding may select from the input data items provided in the XML request message since the integration server may preserve the self-describing properties of these data items. In step 3, the input binding embodies the input arguments as standard CGI (common gateway interface) query parameters and submits the parameters to the web server. In step 4, the web server performs some action in response to these parameters, such as updating a database or retrieving information from a database. In step 5, the web server replies with an HTML document, sending the document back to the integration server via HTTP. In step 6, the integration server applies a WIDL output binding to the HTML document to extract output parameters from the document. In step 7, the integration server generates an XML reply message containing the output parameters and sends the reply message to the client machine. The client machine has therefore transmitted an XML request message, invoked a web site service, and received an XML response message.

FIG. 9 portrays another use of XML RPC to invoke a service that is not a function. In this case the service is a database query. In step 1, the client machine generates and transmits an XML-based request message to the integration server. The integration server extracts arguments from the request message. In step 2, the integration server applies a query template to the arguments to generate a query (such as an SQL query) from the arguments. For example, the query template might appear as follows:

```
SELECT TITLE, SALARY FROM EMPLOYEES
  WHERE YEARHIRED <= % year %
```

If the XML request message contains a data item whose semantic label is "year", and if the data item has a value of "1997", then the resulting query is as follows:

```
SELECT TITLE, SALARY FROM EMPLOYEES
  WHERE YEARHIRED <= 1997
```

In step 3, the integration server submits this query to the database for execute. In step 4, the database executes the query. In step 5, the database returns the results of the query

The results usually take the form of a table. In step **6**, the integration server generates an XML reply message that expresses the results of the query and sends the reply message to the client machine. Hence, the integration server has used the self-describing properties of an XML request message to perform a database query without having to first invoke a function that has been specifically implemented to perform the query. A database query is therefore an example of a service that is not a function.

To use XML RPC to invoke a service, the client machine must identify the service that the request message is intended to invoke. There are two basic ways to accomplish this. Transfer protocols require that the client provide both a destination indicator and a message at the time the client sends the message to a server. One way to identify the target service is to identify the service within the destination indicator. For example, if the transfer protocol is HTTP, the URL (also known as the URI) serves as the destination indicator. In addition to naming the host server, as URLs conventionally do, the URL would also name the service that the client intends to invoke. The following URL illustrates how one may invoke a service named "DoSomething" on the webMethods B2B Integration Server:

http://b2b.companyX.com/invoke/interfaceY/DoSomething

In this example, "b2b.companyX.com" identifies the server that hosts the service, "invoke" informs the server that the server is to invoke a service, "interfaceY" identifies the group of services to which the service belongs, and "DoSomething" identifies the name of the service to invoke from this group.

Another way to identify the target service is to do so within the request message. Since the request message is in XML, there are as many ways to identify the service within the message as there are ways to embed information in an XML document. For example, one might name the target service either in an attribute value or in the content of an element. When the message identifies the target service, the destination identifier need not also do so. The following XML message illustrates identifying the target service within an element attribute:

```
<RPC-REQUEST SERVICE="DoSomething">
  <!-- body of the request message --!>
</RPC-REQUEST>
```

Still another way to accomplish this is to use the element type name of the root element to identify the service, as illustrated in the following example:

```
<DoSomething>
  <!-- body of the request message --!>
</DoSomething>
```

In addition to identifying the target service, the client must create the request message (possibly inserting the target service name into the request message). The conventional RPC mechanism uses stub code to accomplish this. The stub is hard-coded to produce the message that is associated with each function or class-method. A developer generally uses an IDL compiler to generate this code. When the client generates an XML-based RPC message, another option is available to the client: the client may use a template to generate the request message. A template may take any of a variety of forms. In this case, it may take the form of the XML document that is to be generated, except for placeholders within the template that specify where the data items of the service's output arguments are to be placed within the document. An example template follows:

```
<PurchaseOrder>
    <Account>%value acct%</Account>
%loop line-items%
    <LineItem>
        <Model>%value model%</Model>
        <Quantity>%value quantity%</Quantity>
    </LineItem>
%endloop%
</PurchaseOrder>
```

In this example, the template generates a specific XML-based RPC message that represents a purchase order. The '%' symbol delimits placeholders. The placeholders that begin with "% value" name data items that are to be inserted into the document. In the case of "% value acct %" the data item "acct" is the name of an input argument that was provided to the service or function. The "% loop" placeholder identifies a data item that contains a set of data items. The template repeats the contents between the "% loop" and the "% endloop %" placeholders one for each value in this data item. The "% value" placeholders in this data item each name a data item that occurs within a value of the set identified by "% loop". Hence, in this example, the "line-item" data item is a set of data items such that each data item in the set contains both a data item labeled "model" and a data item labeled "quantity".

When the server receives an XML-based RPC request message from the client, the server must identify the target service and it must convert the data items of the request message into a form that is suitable for the target service. If the request message names the target service, the server must first determine the message's document type so that it knows which data item contains the target service. The server must also translate the request message into the arguments that it is to deliver to the target service. The translation mechanism that the server uses may depend on the document type in which the message is expressed.

There are a variety of approaches to determining a message's document type. Several approaches follow:

XML allows a document to identify its document type in the optional document type declaration (DTD) that may appear within the document. The DTD has an optional system identifier and an optional public identifier. These identifiers contain URLs (or URIs) that uniquely identify the document type within the space of all document types. The server could parse the XML message and then extract the document type from the DTD. The XML specification explains this use of the system and public document type identifiers.

A parameter of the transfer protocol could identify the document type. For example, an HTTP request could identify the document type in an HTTP header variable. The target URL (or URI) might itself identify the document type.

The element type name of the root element could identify the document type. The server would be configured to associate element type names with the particular document types.

The server could be hard-coded or configured to accept only messages expressed in a particular document type. This approach would be most useful in servers that used generic encodings, since a single generic encoding (that is, a single document type) would be able to represent any RPC request or reply message.

The service being invoked might be configured to accept only messages of a particular document type. This approach cannot be used with document types that assume responsibility for identifying the service, since the service must be known before the document type can be determined.

The present invention contemplates expressing each XML-based message-encoding as either a generic encoding or a specific encoding. Hybrid encodings are possible, but if any portion of a message is application-specific or industry-specific, the message is itself necessarily application-specific or industry-specific. A generic XML-based encoding uses element type names and attribute names (the labels) to describe the data structures and the data types to which the data items conform. The server uses this information to build the data structures that a generic message depicts. The following XML documents illustrate request and reply messages that are expressed in an XML-based generic encoding:

```
<RPC TYPE="REQUEST">
  <VALUE NAME="accountID" TYPE="int">2001</VALUE>
  <VALUE NAME="zodiacSign">Aquarius</VALUE>
</RPC>
<RPC TYPE="REPLY">
  <VALUE NAME="orderNumber" TYPE="int">38553</VALUE>
  <VALUE NAME="fortune">XML is good for RPC</VALUE>
  <VALUE NAME="balance" TYPE="float">65.00</VALUE>
</RPC>
```

These messages provide a name attribute for every data item. The name attribute associates a semantic label with the data item. The remaining element type names and attributes provide data structure and type information. The messages might also have been expressed in an encoding that lacks the name attributes (or for which the name attributes are optional). Programs using messages expressed in such an encoding would only be able to construct the data structures represented in the messages and would not be able to distinguish among and selectively process the data items without being configured or hard-coded with semantic information about the data items. Thus, it is beneficial for a generic XML-based encoding to include not only labels that provide the data structures and data types of the data items, but to also include labels that semantically label the data items. Absent the semantic labels provided by the name attributes shown in the preceding example messages, the messages would be expressed as follows:

```
<RPC TYPE="REQUEST">
  <VALUE TYPE="int">2001</VALUE>
  <VALUE>Aquarius</VALUE>
</RPC>
<RPC TYPE="REPLY">
  <VALUE TYPE="int">38553</VALUE>
  <VALUE>XML is good for RPC</VALUE>
  <VALUE TYPE="float">65.00</VALUE>
</RPC>
```

Many other XML-based generic encodings are possible. As previously explained IIOP and DCE are already using generic encodings for RPC. XML-based generic encodings according to this invention are unique for the many reasons that have already been ascribed to the class of XML-based messages. However, they are especially unique because unlike IIOP and DCE messages, XML-based messages according to this invention are self-describing and therefore acquire all of the benefits that this application has ascribed to self-describing encodings. In particular, because XML-based generic encodings provide data structure and type information for the individual data items, a client can invoke a remote service on a server without configuring or hard-coding the server to recognize the message types that are specific to that service. The messages themselves contain all of the information necessary to decode the data found in the message into the arguments that the service requires, since the message itself describes the data structures of the arguments.

Notice that it is not necessary for the in-memory representation the client uses for these arguments to be the same representation into which the server reconstructs the arguments, since the server and client platforms may have different representational requirements, and since some data structures have multiple equivalent representations. For example, a client written in the C programming language might represent an array of records (C struct types) as a contiguous sequence of records, while a server written in the Java programming language might represent an array of records as an array of pointers (object references) to records that have been allocated in a possibly non-contiguous manner. Middleware on the server computer would assume responsibility for receiving the message and for converting the message into the representation required by the service being invoked, since the client generally will not (and generally should not) convey information about either a server's platform-specific requirements or a service's language-specific requirements.

A specific XML-based encoding uses element type names and attribute names (the labels) to label the data items according to the semantics that the data items have in the associated application or industry. A specific encoding might also use type labels, but because some of the labels are meaningful only to a particular application or industry, the encoding is not considered to be a generic encoding. The following examples are specific encodings of the above generically encoded messages:

```
<FORTUNE-REQUEST>
  <accountID>2001</accountID>
  <zodiacSign>Aquarius</zodiacSign>
</FORTUNE-REQUEST>
<FORTUNE-RECEIPT>
  <orderNumber>38553</orderNumber>
  <fortune>XML is good for RPC</fortune>
  <balance>65.00</balance>
</FORTUNE-RECEIPT>
```

Note that the message contains no type labels, which prevents a server that receives the message from being able to construct the corresponding data structure unless the server is pre-configured or hard-coded with knowledge of this data structure. For example, without knowing in advance that the contents of the 'orderNumber' element is always an integer, the server would have to represent the contents as a string.

There are many ways in which to express equivalent specific encodings. For example, a data item that occurs as the character data content of an element might instead occur as the value of an attribute, or a data item that occurs as a child of one element type might instead occur as a child of another element type. Data items might also assume different element type names or attribute names in different but equivalent encodings. This flexibility is the primary asset of specific encodings, since it allows people to define encodings that are specific to an application or an industry. An encoding may be designed so that messages expressed in the encoding are readable, writable, and modifiable by in

experts who may not also be software development experts. Such encodings may be designed to directly and succinctly portray the data structures with which the industry experts are familiar.

Implementing RPC using a generic encoding requires certain changes. The W3C provides a standard known as DOM (Document Object Model). DOM is an API for accessing the contents of HTML and XML documents. Many parsers exist that will parse an XML document into a data structure that is navigable via the DOM API. To implement a generic encoding a software developer need only use this API to navigate a generically encoded XML document and to build the data structures identified by the labels found in the document. This application refers to the code that implements this mechanism as a 'codec.' The name derives from a codec's responsibility for encoding and decoding the XML RPC messages.

In an architecture that uses codecs, codecs need only be responsible for converting a set of arguments into a message and for extracting a set of arguments from a message. Codecs need not be responsible for transmitting and receiving these messages. FIG. **10** provides a logical view of an architecture that uses codecs. The diagram depicts the steps involved in sending and receiving messages that are encoded and decoded by codecs on both the client and server machines. Steps **1** and **2** together comprise a unit whereby a codec generates a message from a set of arguments. In this case the arguments are the input arguments of the request. In step **1**, the codec generates an internal representation of the XML request message from the arguments. One might use the W3C DOM APIs to encapsulate this representation. In step **2**, the codec generates a serialized representation of the message. The serialized representation is expressed in the syntax of XML and is suitable for transmission between machines. When the client application wishes to invoke a service via XML RPC, the client machine utilizes the appropriate codec to generate the message and then transmits the message to the server machine.

Steps **3** and **4** together comprise the reverse codec operation. Here a codec extracts a set of arguments from a provided XML message. In this case the arguments are input arguments. In step **3**, the client machine hands the codec an XML message. The codec parses the message to generate an internal representation of the message. In step **4**, the codec navigates this representation to extract the arguments from the message. If the representation exposes itself through DOM APIs, then the codec navigates the DOM APIs to gather this information. The resulting arguments are suitable for use by a programming language function.

Steps **5** and **6** together perform the same operation as steps **1** and **2**, except that in this case the client machine is asking the codec to create an XML message containing output arguments. Likewise, steps **7** and **8** perform the same operation as steps **3** and **4**, except that the client machine is asking the codec to extract output arguments from the message. Because the operations are symmetrical, the same codec may be used on both the client and the server. The net result is that a client application encodes and transmits encoded arguments to a server application, which decodes the arguments and invokes a service using the arguments as inputs. When the service completes, the server application encodes the service's output arguments and transmits the encoded output arguments to the client application, which then decodes and utilizes the output arguments.

A codec must be able to generate an XML message from a set of input arguments. This requires that the codec acquire knowledge of the type structures of the arguments. It is easiest to imagine implementing the encoding procedure for languages that provide run-time type information, such as C++, Java, and Smalltalk. The codec need only examine the type information that the language associates with the arguments and use string concatenation to build the reply XML message in accordance with this type information. Implementing the encoding procedure for other languages requires the type information to be configured or hard-coded in either the codec or the software that asks the codec to perform the encoding. Conventional RPC mechanisms use the configuring or hardcoding approach. A service could provide the type information for its input and output arguments via an interface repository (abbreviated 'IR' in the OMG CORBA terminology).

There are several approaches to implementing XML-based specific encodings. One approach is analogous to the implementation of generic encodings. Just as for generic encodings in which one writes a codec that is specifically designed to recognize and generate the particular generic encoding, in this approach to implementing specific encodings, one writes a codec that is hard-coded to recognize and generate a particular specific encoding. As with the implementation of generic encodings, one could use the DOM APIs and run-time type information to accomplish this. This approach has the drawback that distinct specific encodings require distinct software code, which generally inclines developers to write software to process only a single encoding or a small number of encodings.

In another approach to implementing XML-based specific encodings, a single implementation is capable of processing any encoding that it is configured to recognize. The approach may also be used to implement generic encodings, although the approach requires that the implementation be configured to recognize each particular kind of message that will be expressed in the generic encoding. In generic encodings, the kind of message varies according to the data structures that are represented in the different messages. This approach requires the use of binding technology that the inventors have defined, and which is described in more detail in the publications cited above herein.

FIG. **11** depicts the architecture of the binding approach to implementing XML-based specific encodings. The approach distinguishes client behavior from server behavior. The client machine may implement the client behavior regardless of the server implementation, and the server machine may implement the server behavior regardless of the client implementation. The figure depicts a scenario in which both client machine and server machine exhibit binding behavior. In step **1**, the client application applies a template to a set of input arguments. According to the example, the result is an XML request message. The client machine transmits this message to the server machine. In step **2**, the server machine parses the XML message into an internal representation. In step **3**, the server applies a binding to this internal representation to yield a set of input arguments and then invokes a service with these input arguments. In step **4**, the service completes and returns a set of output arguments. The server machine applies a template to these output arguments to generate an XML reply message and then transmits this message to the client machine. In step **5**, the client machine parses the reply message into an internal representation. In step **6**, the client machine applies a binding to this internal representation to yield a set of output arguments. These output arguments are returned to the application as output arguments of the invoked service.

The integration server is capable of associating each encoding with a particular codec or binding. (The serv

also accomplish this by associating each service with a particular codec or binding or by allowing each service to select the codec or binding that applies.) FIG. 12 depicts this facility. When the server receives a request message, it applies the appropriate codec or binding to the message to extract the input arguments from the message. The server then invokes the service using these data structures as arguments. When the service completes and returns output arguments, the server applies a template to these output arguments to generate the XML message that is to be returned to the client. A program that implements this architecture is capable of servicing XML RPC messages expressed in a variety of encodings. FIG. 13 illustrates the architecture's benefits. Client applications can be written to use different message encodings. The integration server is capable of invoking the same service in response to receiving messages expressed in any of these encodings. One service becomes available to a variety of clients designed for use with differing XML document types.

This application has so far described two mechanisms for generating XML-based RPC messages. One is to use program code that is specifically designed to generate only a particular encoding. Another is to use the template mechanism. The approaches are applicable to both generating request messages on the client side and to generating reply messages on the server side. This application has also described two mechanisms for extracting data items from XML-based RPC messages. One is to use a codec that is specifically designed to extract data items for a particular encoding. This is the mechanism that is most suitable for implementation based on XML APIs such as the W3C DOM API. The other is to use the binding mechanism, the detailed description of which has been left to documents that are included above in this application by reference. The approaches for extracting data items are applicable both to receiving request messages on the server side and to receiving reply messages on the client side. Hence, several techniques have been described by which one may implement XML RPC.

The present inventors have identified a simplification to the RPC mechanism that may be applied to any kind of encoding, including both specific and generic XML-based encodings. Conventional implementations of RPC middleware use code known as stubs or skeletons to translate between the middleware-specific representation of a message and the function signature required by the service. Conventionally, if a message includes three distinct record arguments that are to be input to the service, the service's signature may have three distinct record arguments. The following Java code illustrates such a service signature:

```
EmployeeID addEmployee(Name n, Address a, JobType
    t)
```

In a conventional middleware solution, a developer would use a tool known as an Interface Definition Language (IDL) compiler to generate stub or skeleton code that invokes this service by parceling out the individual message arguments into individual service arguments. The same stub or skeleton code would be responsible for converting the output EmployeeID argument into a data item of the reply message.

The present inventors contemplate the simplification whereby all services have the same signature except for possibly the name of the service. This signature would take as input a single software object that contains all of the input data items and would return as output a single software object that contains all of the output arguments. The inventors have defined an object known as a Values object for this purpose. A Values object is a hash table of named software objects where the object names serve as the hash table keys. The object names serve both as the key by which the objects are hashed and as labels for the software objects. The object names may actually be labels found in the message, but they need not be so. The objects of the hash table may represent lexical types such as integers, floats, and strings. They may also represent structured objects such as arrays, lists of arbitrary objects, and records. Records may be represented within a Values object as another Values object. Values objects are well-suited for representing records, since records uniquely name their constituent fields, making them good hash keys.

The Values object serves as the single input object of a service and as the single output object of a service. The previous example service could then be expressed as follows:

```
Values addEmployee(Values inputs)
```

The Values object provided as an argument would contain at least three objects one object for each of Name, Address, and JobType. However, the Values object would represent the Name, Address, and JobType structures as nested Values objects. That is, a Values object may represent records internally as other values objects. The fields that comprise the Name, Address, and JobType structures would then each occur as an entry in the associated nested Values object. The service may but need not return the Values object instance that is provided as an input argument.

This simplification benefits the software developer by eliminating the need for the developer to run an IDL compiler and by speeding up the RPC mechanism. The RPC mechanism can be implemented so that it is faster than the conventional mechanism because it allows the developer to eliminate the need to translate between the middleware-specific data representation and the representation against which the developer codes. Furthermore, it allows the message to contain values that the service does not recognize or that an integration server provides for the service's optional use. For example, the message's encoding may evolve so that it contains more information than the service was originally designed to accept. This mechanism allows the service to continue to function with the new encoding by utilizing only the information that existed in the pervious version of the encoding. As another example, the integration server might include within the input Values object an entry that identifies the encoding in which the message was expressed.

One may improve this approach by adding a state parameter to each service signature so that the integration server can be responsible for managing the persistence of state. The service then manages the state by modifying this parameter. The integration server associates each client session with a particular state object and hands the state object to a service every time the client invokes the service. The state object might be a Values object or an object specifically designed to represent session state. The following code illustrates how this improvement could be expressed in the service signature:

```
Values addEmployee(Session session, Values inputs)
```

One of the apparent drawbacks of the single input object and single output object approach is that it requires the software developer to translate between the integration server representation of the data and the representation that the developer requires, should the developer require a different representation. The inventors mitigate this drawback by allowing the developer to use a special IDL compiler that

generates utility software. The developer can then hand the input Values object to the generated software to have the developer-required data structures automatically generated for the developer. For example, the IDL compiler might generate the following Java code to convert an input values object into a native data structure:

```
class Address {
    String street1;
    String street2;
    String city;
    String state;
    String zip;
    Address (Values inputs) {
        street1 = inputs.get("street1");
        street2 = inputs.get("street2");
        city = inputs.get("city");
        state = inputs.get("state");
        zip = inputs.get("zip");
    }
}
```

The XML RPC improvements provide many benefits, including those already described in this section, and including the following improvements over conventional RPC mechanisms:

- The RPC mechanism operates statelessly over HTTP, requiring significantly less complex software is required to implement RPC over HTTP. The session-based state objects described in this section do not provide state for the RPC transfer protocol, but rather only provide state for the services that use the protocol, so that this state mechanism is not in violation of the requirement for stateless RPC. In any case, the session mechanism is not a mandatory feature of XML RPC.
- The RPC mechanism does not support bi-directional communications, requiring significantly less complex client software to implement RPC over HTTP. Only solutions that absolutely require bi-directional communications would suffer added complexity, and they might do so by putting an integration server on the client computer so that the client may also receive RPC messages.
- The RPC mechanism exhibits sufficient simplicity that two programs could engage in RPC communications over a network without requiring software from a single vendor to be installed on both of the computers that are running the programs. The client of a server that supports XML RPC only needs an XML parser and an HTTP stack. An implementation of the DOM API would also be preferable, but it would not be necessary. The client program can concatenate strings to generate XML and then submit the XML to the server via the standard HTTP GET or POST methods, and the client could parse the reply document that HTTP returns using the now-ubiquitous XML parser. The client would then use the DOM APIs to navigate the document to extract the data that the client requires. As this example demonstrates, in stark contrast to conventional RPC mechanisms, a client can use XML RPC without using any vendor client XML RPC software at all.

The 80/20 RPC Encoding

As previously stated, conventional RPC protocols such as IIOP and DCE use generic encodings that are not XML-based. These generic encodings essentially comprise the union of features required by the individual programming languages and the different software platforms. Consequently, conventional encodings are complex. For example, these encodings distinguish among a wide variety of data types, such as short integers, long integers, enumerations, floating-point numbers in various precisions, fixed-point numbers, unions, strings, and wide strings. In addition, these protocols must define the endianness and lengths of data items, including whether arrays are represented row-first or column-first. The protocols also support the communication of programming language exceptions, error information, synchronity, network-aware object references, and other sophisticated information. Consequently, the developers of conventional RPC solutions are almost universally reliant on software tools that completely hide the encodings from the developer.

One might easily design generic XML-based encodings that suffer from the same complexities. The RPC community has operated under the assumption that it is the encoding's responsibility to provide the superset of the features available to programming languages and software platforms, but the present assignees have created an RPC mechanism that allows varied programs to communicate via a simple generic encoding. This encoding is an XML-based encoding, and the assignees refer to it as the 80/20 RPC encoding. The encoding is based on the principle that an encoding should only provide the functionality required by 80% of the solutions that might use the encoding and that the remaining 20% should be relegated to protocols that layer on top of the encoding. The 80/20 RPC encoding provides about 80% of the functionality of conventional RPC encodings for less than 20% of the complexity.

It is not new to the RPC community to layer protocols on top of RPC encodings. The IDispatch interface is an example of such a layered protocol. However, the 80/20 RPC encoding provides a different partitioning of functionality among the layers. Instead of putting the self-describing layer on top of the layer that supports rich data types, the 80/20 RPC encoding puts the self-describing layer at the bottom and allows applications to build sophisticated data types on top of this layer. For example, DCE support unions while the 80/20 RPC encoding does not, but one may easily build unions on top of the 80/20 RPC encoding in the manner that IDispatch is built on DCE. As another example, consider that the 80/20 RPC does not natively support the referencing of network objects and asynchronous communications with those objects. Suppose an application requires this functionality, and suppose the application is using HTTP as the transport protocol. The application may provide asynchronous communications on top of the 80/20 RPC encoding by implementing the RPC in both directions by installing on both the client and server machines an integration server that uses the 80/20 RPC encoding over HTTP.

The 80/20 RPC encoding provides another significant benefit over conventional RPC encodings. This benefit is a consequence of the encoding being both text-based and flexible in its representation of data items. In the encoding, one uses text to represent the values of all data items. For example, an integer would be expressed as a sequence of human-readable decimal digits rather than the two, four, or eight binary bytes that IIOP or DCE would require. One may therefore buffer an integer with preceding zeroes, should this be a readability requirement. One may also place a floating-point value in a data item that expects an integer and have it automatically rounded to the nearest integer. Likewise, one may suffix numbers with a units indicator term (as in "30 barrels") and have the term ignored.

RPC mechanisms have heretofore not included such facilities for automated typecasting and human-read

accommodation. The 80/20 RPC approach inclines developers to design more human-accessible messages and further supports the philosophy that protocol layers above the encoding should assume responsibility for rich typing. It is expected that higher layers will grow unbounded in sophistication to support the endlessly rich variety of human-readable types.

In order for an encoding to provide this simplicity of expression and this repartitioning of functionality, the inventors had to select the domain of applicability for which the encoding would provide 80% of the required functionality. The inventors selected the domain of business-to-business integration over the Internet. This domain focuses on connecting applications to applications over the Internet, and the applications are typically self-contained except for their need to engage in business transactions with other businesses. The 80/20 RPC encoding is ideal for this usage and reflects the experience that the inventors have in this area.

The 80/20 RPC encoding is independent of the syntax used to express the encoding, but the inventors have implemented it in XML, so the present application expresses the encoding in XML. The encoding requires only six distinct labels, and the XML-based syntax expresses these labels as XML element type names. Each label corresponds to a type of data item. Each data item may also have additional descriptive information. but this information is optional. The XML-based representation expresses this optional information using XML attributes. The attributes available to any element type vary according to element type. The six element types and their attributes follow:

- VALUE—This element represents a data item that is a lexical unit, and it has an optional TYPE attribute. Lexical units include integers, floating-point numbers, strings, boolean values, etc. They generally correspond to the primitive data types of most programming languages. The value of the optional TYPE attribute names the lexical type. The inventor's implementation uses lexical type names such as Integer, Long, Float, and String. VALUE elements that lack the TYPE attribute are assumed to be strings. Variations of the encoding might assume another type in the absence of the TYPE attribute, such as the integer type.
- RECORD—This element contains (holds as children) a set of one or more named elements of any type. Each child is required to have a NAME attribute, and the value of each NAME attribute must be unique among the values of the NAME attributes of all children of the containing RECORD element. The NAME attribute serves as a means for attaching semantic labels to the data items, but semantic labels are not necessary, as one may also identify elements according to their position within the RECORD element. However, as described above, semantic labels add value to the message. The RECORD element is always the root element of an 80/20 RPC message.
- LIST This element contains a set of elements. The elements contained in a LIST do not have names; or if they do have names, the names are ignored.
- OBJECT—This element references another element by object ID, and it has a mandatory REFERENCE attribute. Any element in the 80/20 RPC encoding may have an attribute of type ID. The value of the REFERENCE attribute is the value of the ID attribute of another element, and this other element is known as the element that the OBJECT element references. An OBJECT element serves as a placeholder for the element it references. An OBJECT element is equivalent to a programming language pointer or object reference to a software object. The element therefore provides a mechanism by which one software object may be included within two or more other objects. However, the element only serves as a suggestion to the server that when the server converts the message into software objects, the server should create only one instance of the referenced object.
- NULL—This element represents a data item for which no data is provided. It is equivalent to a null-pointer in the case where it replaces objects that a programming language might represent using a pointer. It is equivalent to NULL or NIL in programming languages that are capable of distinguishing between assigned and unassigned variables of primitive data types. In the context of a containing RECORD, LIST, or ARRAY element (where the ARRAY element is not of a string type), the NULL element provides a simple way to disambiguate between the absence of a value and the occurrence of a null pointer. In the context of a containing VALUE or ARRAY element (where the ARRAY element is of a primitive type), the NULL element provides a simple way to disambiguate between the absence of a value and presence of either the empty string or a numeric value of zero.
- ARRAY—This element represents a single or multi-dimensional array of elements, and it has an optional DIMENSION attribute and an optional TYPE attribute. The DIMENSION attribute indicates the dimensionality or depth of the array, and the TYPE attribute indicates the type of data found in the array. Both of these attributes are optional, since the same information can be acquired by examining the content of the ARRAY element, but their presence facilitates the efficient processing of the message by allowing the server to allocate the required memory prior to processing the content of the ARRAY element. All immediate children of an ARRAY element must be of the same element type, except that NULL elements are always valid.

Any element may have a NAME attribute. The value of the NAME attribute uniquely identifies the element within a containing RECORD element and is only required when the element is a child of a RECORD element. No two NAME attribute values may be identical within the scope of the same parent RECORD element. The purpose of the NAME attribute is to give semantic labels to the data items in a RECORD element. The NAME attribute is ignored when it occurs in any position other than as the attribute of a child of a RECORD element.

Any element may also have an ID attribute. The value of the ID attribute uniquely identifies the element within the 80/20 RPC message. No two ID attribute values may be identical within the scope of the same RPC message. The purpose of the ID attribute is to allow a single data item to be contained in multiple data items. Without the ID attribute and the OBJECT element, the RPC messages would only be able to express tree data structures. With the ID attribute and the OBJECT element, the RPC messages can express arbitrary directed graphs, including cyclic graphs.

Notice that the preferred embodiment of 80/20 RPC associates at least one type label and at least one semantic label with every data item. Data items are represented by XML elements, and the element type name of an XML element serves as a type label, except for the OBJECT element name. However, each OBJECT element references another element that does have an associated type label.

type label associated with the referenced element is therefore also a type label of the OBJECT element. The VALUE and ARRAY elements may have multiple associated type labels that together define the data type of the associated data item.

Likewise, every argument and every data item contained in every argument is associated with at least one semantic label. The semantic label of a data item that resides in a RECORD element is the value of the NAME attribute that is present on the element. The semantic label of any data item not already having a semantic label is the semantic label associated with nearest ancestor RECORD element of the element representing the data item. For example, if a list has semantic label X within a record, then the data items contained in the list each have semantic label X. To be more concrete, if an array of products is labeled “ProductList,” then every product in the array is labeled “ProductList.” Notice that the semantic label attached to an OBJECT element is never the semantic label of the data item that the element references.

The 80/20 RPC relegates to the transfer protocol all responsibility for identifying the service that is to be invoked, so no element in the encoding actually names the target service. Consider the following Java classes:

```
class Person {
    String name;
    Integer age;
    Address address;
    Job[ ] jobHistory;
    Vector credentials; // Degrees and Awards
}
class Address {
    String[ ] streetLines;
    String zip;
}
class Degree {
    String degree;
    String school;
}
class Award {
    String award;
    String date;
}
class Job {
    String company;
    Address address;
    String title;
}
```

Suppose a client wishes to invoke a service having the following Java signature:

Job addEmployee(Person p)

The service adds an employee to the employee database, determines an appropriate job for the employee and returns the description of that job. When the client invokes this service the client puts the service name somewhere in the transfer protocol header. In the case of HTTP, it is economical to name the service within the URL or URI. The client then sends an 80/20 RPC message to the server.

Here is an example of a message that the client might send for this particular service:

```
<RECORD>
    <VALUE NAME=“name”>Joe Shmoe</VALUE>
    <VALUE NAME=“age” TYPE=“Integer”>30</VALUE>
```

-continued

```
    <RECORD NAME=“address” ID=“A1”>
        <ARRAY NAME=“streetLines”>
            <VALUE>1234 Someplace Court</VALUE>
        </ARRAY>
        <VALUE NAME=“zip”>112233</VALUE>
    </RECORD>
    <ARRAY NAME=“jobHistory” TYPE=“RECORD”>
        <RECORD>
            <VALUE NAME=“company”>Big Deal Corp.
                </VALUE>
            <RECORD NAME=“address”>
                <ARRAY NAME=“streetLines”>
                    <VALUE>99 Business Lane</VALUE>
                    <VALUE>Suite 1010</VALUE>
                </ARRAY>
                <VALUE NAME=“zip”>112233</VALUE>
            </RECORD>
            <VALUE NAME=“title”>Worker</VALUE>
        </RECORD>
        <RECORD>
            <VALUE NAME=“company”>Shmoe's Consulting
                </VALUE>
            <OBJECT NAME=“address” REFERENCE=“A1”/>
            <VALUE NAME=“title”>Owner</VALUE>
        </RECORD>
    </ARRAY>
    <LIST NAME=“credentials”>
        <RECORD>
            <VALUE NAME=“degree”>Thinking</VALUE>
            <VALUE NAME=“school”>MIT</VALUE>
        </RECORD>
        <RECORD>
            <VALUE NAME=“award”>Big Shot</VALUE>
            <VALUE NAME=“date”>1/1/98</VALUE>
        </RECORD>
        <VALUE>Nobel Prize for Physics</VALUE>
    </LIST>
</RECORD>
```

Notice a few characteristics of this message. The message describes a person and includes the person's address and the address of the person's various employers. At one point the person worked for himself and hence his employer's address is identical to his home address. The message represents this by having the Job entry for this employment period reference the same object that describes the person's home address. The message could have duplicated the object in order to make the message conform strictly to a tree, but this makes inefficient use of memory. Also, if the address is found to be in error or if the zip code subsequently changes, the resulting data structure can be corrected by changing one RECORD instead of changing two.

Also notice that the LIST element may contain a mix of element types. Each programming language will have to find a suitable representation for this notion. The LIST element is intended to be the mechanism by which objects of arbitrary types may be collected together into a single list of arbitrary length. For example, this construct is useful for representing instances of Java's Vector class. Programming languages such as Visual Basic might represent LIST elements as VARIANT variables. Languages such as C and C++ might represent LIST elements using special data types that provide information about the type of each element.

The above example does not illustrate multi-dimensional arrays. An array of dimension two or greater is represented by nesting ARRAY elements within ARRAY elements. The depth of the nesting equals the dimensionality minus one. This approach takes advantage of the fact that the encoding is expressed in XML and is not otherwise available to conventional binary representations of arrays. A conventional binary representation of a multi-dimensional array

defined by a set of dimension sizes, and the size of each dimension except for the last dimension must be known. For example, expressed in XML, the analogue of a conventional representation might appear as follows for a two-dimensional array:

```
<BINARY-LIKE-ARRAY DIMENSION-SIZES="2,3">
  <VALUE>this is 0.0</VALUE>
  <VALUE>this is 0.1</VALUE>
  <VALUE>this is 0.2</VALUE>
  <VALUE>this is 1.0</VALUE>
  <VALUE>this is 1.1</VALUE>
  <VALUE>this is 1.2</VALUE>
</BINARY-LIKE-ARRAY>
```

The 80/20 RPC expresses the above array as follows:

```
<ARRAY DIMENSION="2">
  <ARRAY>
    <VALUE>this is 0,0</VALUE>
    <VALUE>this is 0,1</VALUE>
    <VALUE>this is 0,2</VALUE>
  </ARRAY>
  <ARRAY>
    <VALUE>this is 1,0</VALUE>
    <VALUE>this is 1,1</VALUE>
    <VALUE>this is 1,2</VALUE>
  </ARRAY>
</ARRAY>
```

Consider another example. Here is the analogue of a three-dimensional binary array:

```
<BINARY-LIKE-ARRAY DIMENSION-SIZES="2,2,
  2">
  <VALUE> 0, 0, 0</VALUE>
  <VALUE> 0, 0, 1</VALUE>
  <VALUE> 0, 1, 0</VALUE>
  <VALUE> 0, 1, 1</VALUE>
  <VALUE> 1, 0, 0</VALUE>
  <VALUE> 1, 0, 1</VALUE>
  <VALUE> 1, 1, 0</VALUE>
  <VALUE> 1, 1, 1</VALUE>
</BINARY-LIKE-ARRAY>
```

The 80/20 RPC expresses the above array as follows:

```
<ARRAY DIMENSION="3">
  <ARRAY>
    <ARRAY>
      <VALUE> 0, 0, 0 </VALUE>
      <VALUE> 0, 0, 1 </VALUE>
    </ARRAY>
    <ARRAY>
      <VALUE> 0, 1, 0 </VALUE>
      <VALUE> 0, 1, 1 </VALUE>
    </ARRAY>
  </ARRAY>
  <ARRAY>
    <ARRAY>
      <VALUE> 1, 0, 0 </VALUE>
      <VALUE> 1, 0, 1 </VALUE>
    </ARRAY>
    <ARRAY>
      <VALUE> 1, 1, 0 </VALUE>
      <VALUE> 1, 1, 1 </VALUE>
    </ARRAY>
  </ARRAY>
</ARRAY>
```

This approach to representing arrays allows an ARRAY element to represent an array of pointers (or object references) to arrays. Thus, it is not necessary to require that the array entries for a given dimension of the array be represented internally with identical array sizes. Also, the ARRAY element nesting approach provides a more natural way for humans to view the structure of the array within the XML representation.

Reply messages of the 80/20 RPC are expressed in the same encoding as the request messages. Here is an example of a reply message for the above request message:

```
<RECORD>
  <VALUE NAME="company">New Employer Corp.</VALUE>
  <RECORD NAME="address">
    <ARRAY NAME="streetLines">
      <VALUE>5 New Company Plaza</VALUE>
      <VALUE>Suite 4321</VALUE>
    </ARRAY>
    <VALUE NAME="zip">10203</VALUE>
  </RECORD>
  <VALUE NAME="title">Manager</VALUE>
</RECORD>
```

In the preceding examples, both the client and the server receive RECORD elements that correspond to the Job class portrayed above. The client and the server may chose distinct representations for this RECORD element. One side might represent it as a Values object, while the other side might construct a more dedicated structure to hold the object, such as a C struct instance. The inventors have implemented an extension to the RECORD element whereby a CLASS attribute may be attached to the element to identify the Java class that the RECORD represents. This extension serves as a suggestion for Java implementations so that a Java implementation may represent the element using an instance of the named class. The above reply message might therefore be expressed as follows:

```
<RECORD CLASS="Job">
  <VALUE NAME="company">New Employer Corp.</VALUE>
  <RECORD NAME="address">
    <ARRAY NAME="streetLines">
      <VALUE>5 New Company Plaza</VALUE>
      <VALUE>Suite 4321</VALUE>
    </ARRAY>
    <VALUE NAME="zip">10203</VALUE>
  </RECORD>
  <VALUE NAME="title">Manager</VALUE>
</RECORD>
```

Non-Java implementations could chose to ignore this attribute, or they could use the attribute to perform equivalent behavior. For example, a C++ or Smalltalk implementation could construct a class instance for a class having this name. The approach to naming RECORD element class could also have been applied to the credential records as follows:

```
<RECORD CLASS="Degree">
  <VALUE NAME="degree">Thinking</VALUE>
  <VALUE NAME="school">MIT</VALUE>
</RECORD>
<RECORD CLASS="Award">
  <VALUE NAME="award">Big Shot</VALUE>
  <VALUE NAME="date">1/1/98</VALUE>
</RECORD>
```

There are several functionally equivalent variations of this RPC encoding. One might select one variation over another according to one's preferences for how readable me

are when expressed in the encodings. A particular variation of the above-described 80/20 RPC encoding separates numeric values and string values into separate elements. For example, one might use the VALUE element type exclusively for string values while using a NUMBER element type for numeric values. The NUMBER element type could support a TYPE attribute to distinguish among types. Another variation assigns each lexical type to its own element type. Such an encoding might support the element types STRING, INTEGER, FLOAT, etc.

There is also value in an encoding that does not distinguish among lexical data types at all. In this encoding all lexical data types are expressed as strings. For example, the VALUE element would be used to express strings and the element would not support the optional TYPE attribute. No other element types would exist to differentiate among lexical types. This approach moves the encoding further in the direction of relegating type information to software layers that reside above the encoding layer. The servers and clients would take responsibility for assigning different programming-language specific data types to the individual data items. Such software could accomplish this by associating each service with a specification for that service, where the specification could be given in an interface specification expressed in some Interface Definition Language (IDL). Alternatively, application-specific code could impose the data types. This approach has the benefit that it frees users to choose more human-readable encodings, although it does so at the expense of additional complexity in the software that must interpret the messages.

It is also possible define a variation of the 80/20 RPC encoding such that the variation names the service-to-invoke within the RPC request message instead of naming this service via the transfer protocol. The service name might be provided as the value of an XML attribute, for example. The two approaches are functionally equivalent, except that if the message names the target service, the server must first parse the message to identify the service, which introduces potential inefficiencies. For example, should the client issuing the request not have access rights to use the service, the server would still have to parse message before denying access. Were the service named via the transfer protocol, the server could have spared the clock cycles and memory necessary to parse the message by ignoring the message and sending the client an error response instead. However, messages that name the target service have the advantage of being complete, so that a user examining the message can readily determine the service for which the message is intended.

The following example illustrates naming the service-to-invoke by using an attribute of the containing RECORD message:

```
<RECORD SERVICE="addEmployee">
  <VALUE NAME="name">Joe Shmoe</VALUE>
  . . .
</RECORD>
```

In another variation of the 80/20 RPC, one might reserve a special element type, such as the RPC element type, for this purpose. This produces a cleaner solution, since the SERVICE attribute is only meaningful on the outermost element of the message. All request messages would then be expressed in terms of this element type. The same element type might also be used to enclose reply messages. In this case it would be useful for the user to have an indication of whether a given message is a request or a reply message. This could be accomplished via a TYPE attribute on this outermost element. The resulting encoding yields messages such as those presented earlier in this application:

```
<RPC TYPE="REQUEST">
  <VALUE NAME="accountID" TYPE="int">2001</
    VALUE>
  <VALUE NAME="zodiacSign">Aquarius</VALUE>
</RPC>
<RPC TYPE="REPLY">
  <VALUE NAME="orderNumber"
    TYPE="int">38553</VALUE>
  <VALUE NAME="fortune">XML is good for RPC</
    VALUE>
  <VALUE NAME="balance" TYPE="float">65.00</
    VALUE>
</RPC>
```

The 80/20 RPC encoding has been expressed here in the syntax of XML, but the value of the encoding is in the data types and the data structures that the encoding uses to represent RPC messages. The encoding could easily be expressed in other syntaxes, including proprietary syntaxes. Messages expressed in the encoding are capable of representing the programming language data types that are most commonly used for business-to-business integration over the Internet. The simplicity and self-describing nature of the encoding lends the encoding the following benefit over conventional RPC encodings:

The 80/20 RPC encoding is simpler than IIOP and DCE. By containing only the features that are necessary for business-to-business electronic commerce over the Internet, it idealizes the encoding for this domain of applications. Furthermore, the encoding's simplicity maximizes the number of software developers that are qualified to produce solutions with the encoding and minimizes the time required to produce each solution.

Important Features

This specification has portrayed a flexible mechanism for implementing RPC. There are a wide variety of ways to implement the invention. One may classify each implementation according to the particular combination of features that the implementation exhibits. This section of the specification itemizes some of these important features. For each feature itemized here, the section also lists some of the modes in which the feature may be expressed in an implementation.

Encoding syntax. Although the specification has focused on using XML as the syntax, the invention is not limited to expression in the XML syntax. The self-describing properties of the message encoding are inventive properties. Hence, one might also express the invention using a binary encoding, a text-based encoding, and an encoding in a markup language other than XML.

Transfer protocol. The specification has focused on using network transfer protocols such as HTTP, SMTP, and FTP, but it is possible to express the invention in other protocols. For example, one might define a proprietary protocol for this purpose. The definition of protocol used by this specification also encompasses the mechanisms by which virtual machines communicate and by which processes in separate address spaces communicate. In particular, the transfer protocol covers many IPCs (Inter-Process Communications).

Service invoked. The specification describes the invention in terms of the services that the invention may be used to invoke. The term 'service' has been described in an abstract way. Services include functions (including procedures and class methods), web sites, and

bases. They also may include middleware platforms, enterprise applications, and any of a myriad set of application protocols.

Argument labels. The specification has described labeling a message argument with a type label. One may label a single argument in this manner, two or more arguments in this manner, or all arguments in this manner. In addition, one may associate a semantic label with arguments having type labels. A semantic label might exist for one of these arguments, for two or more of these arguments, or for all of these arguments.

Constituent labels. The specification has described labeling a data item with a semantic label. In particular, it has described labeling a data item that is contained within an argument. An argument might contain multiple data items. One might label a single such data item, two or more such data items, or all such data items. Furthermore, one might associate a type label with a data item already having an associated semantic label. A type label might exist for one of these data items, for two or more of these data items, or for all of these data items.

Label constraints. As described above, various schemes exist for labeling arguments, and various schemes exist for labeling constituent data items. One might also associate these labeling schemes. For example, an encoding might allow argument labels without regard to whether any constituents of the argument are labeled, and one might allow a constituent of an argument to be labeled without regard to whether the argument is labeled. An encoding might also establish constraints between these labeling schemes. For example, an encoding might require that all constituents of a labeled argument also be labeled. Such constraints add value by allowing an application to optimize for a particular scheme. Requiring that all data items (including arguments) always have both type labels and semantic labels eliminates the need for applications to maintain type and semantic information about the messages it recognizes.

The foregoing discussion of the invention has been presented for purposes of illustration and description. The foregoing is not intended to limit the invention to the form or forms disclosed herein. Although the description of the invention has included description of one or more embodiments and certain variations and modifications, other variations and modifications are within the scope of the invention, e.g. as may be within the skill and knowledge of those in the art, after understanding the present disclosure. It is intended to obtain rights which include alternative embodiments to the extent permitted, including alternate, interchangeable and/or equivalent structures, functions, ranges or steps to those claimed, whether or not such alternate, interchangeable and/or equivalent structures, functions, ranges or steps are disclosed herein, and without intending to publicly dedicate any patentable subject matter.

What is claimed is:

1. A method of invoking a service at a first machine from a second machine, said method comprising the steps of:

generating a service invocation request message at said second machine in compliance with a markup language-based message encoding wherein said message includes elements representing data items of at least one argument and associated with type labels selected from a group of type labels, said group including at least an array type label indicating that the corresponding element is an array element representing an n-dimensional array containing a plurality of data items, where n is an integer and n≧1, said message including at least one element associated with said array type label and representing a multi-level nested array where each element nesting level corresponds to a respective dimension of said array; and

transmitting said service invocation request message from said second machine.

2. A method of invoking a service at a first machine, comprising the steps of:

receiving at said first machine a service invocation request message generated at a second machine in compliance with a markup language-based message encoding wherein said message includes elements representing data items of at least one argument and associated with type labels selected from a group of type labels, said group including at least an array type label indicating that the corresponding element is an array element representing an n-dimensional array containing a plurality of data items, where n is an integer and n≧1, said message including at least one element associated with said array type label and representing a multi-level nested array where each element nesting level corresponds to a respective dimension of said array; and

invoking said service in response to said message.

3. A method of invoking a service at a first machine, said method comprising the steps of:

receiving at said first machine a service invocation request;

invoking said service in response to said request; and

transmitting from said first machine a service invocation reply message in compliance with a markup language-based message encoding wherein said message includes elements representing data items of at least one argument and associated with type labels selected from a group of type labels, said group including at least an array type label indicating that the corresponding element is an array element representing an n-dimensional array containing a plurality of data items, where n is an integer and n≧1, said message including at least one element associated with said array type label and representing a multi-level nested array where each element nesting level corresponds to a respective dimension of said array; and

transmitting said service invocation reply message from said second machine.

4. A method of invoking a service at a first machine, said method comprising the steps of:

transmitting a service invocation request from a second machine; and

receiving at said second machine a service invocation reply message in compliance with a markup language-based message encoding wherein said message includes elements representing data items of at least one argument and associated with type labels selected from a group of type labels, said group including at least an array type label indicating that the corresponding element is an array element representing an n-dimensional array containing a plurality of data items, where n is an integer and n≧1, said message including at least one element associated with said array type label and representing a multi-level nested array where each element nesting level corresponds to a respective dimension of said array.

5. A method of invoking a service at a first machine from a second machine, said method comprising the steps of:

R-845

generating a service invocation request message at said second machine in compliance with a mark-up language-based message encoding wherein said message includes elements representing data items of at least one argument and associated with type labels selected from a group including at least an array type label indicating that the corresponding element is an array element representing an n-dimensional array containing a plurality of data items, where n is an integer and $n \geqq 1$, said request message including at least one element associated with said array type label and representing an array of dimension n and further including an array label associated with said at least one element and requiring that all data items represented within said array have the same type as one another; and

transmitting said service invocation request message from said second machine.

6. A method of invoking a service at a first machine, comprising the steps of:

receiving at said first machine a service invocation request message generated at a second machine in compliance with a markup language-based message encoding wherein said message includes elements representing data items of at least one argument and associated with type labels selected from a group including at least an array type label indicating that the corresponding element is an array element representing an n-dimensional array containing a plurality of data items, where n is an integer and $n \geqq 1$, said request message including at least one element associated with said array type label and representing an array of dimension n and further including an array label associated with said at least one element and requiring that all data items represented within said array have the same type as one another; and

invoking said service in response to said message.

7. A method of invoking a service at a first machine, said method comprising the steps of:

receiving at said first machine a service invocation request;

invoking said service in response to said request; and

transmitting from said first machine a service invocation reply message in compliance with a markup language-based message encoding wherein said message includes elements representing data items of at least one argument and associated with type labels selected from a group including at least an array type label indicating that the corresponding element is an array element representing an n-dimensional array containing a plurality of data items, where n is an integer and $n \geqq 1$, said reply message including at least one element associated with said army type label and representing an array of dimension n and further including an array label associated with said at least one element and requiring that all data items represented within said array have the same type as one another; and

transmitting said service invocation reply message from said second machine.

8. A method of invoking a service at a first machine, said method comprising the steps of:

transmitting a service invocation request from a second machine; and

receiving at said second machine a service invocation reply message in compliance with a markup language-based message encoding wherein said message includes elements representing data items of at least one argument and associated with type labels selected from a group including at least an array type label indicating that the corresponding element is an array element representing an n-dimensional array containing a plurality of data items, where n is an integer and $n \geqq 1$, said reply message including at least one element associated with said array type label and representing an array of dimension n and further including an array label associated with said at least one element and requiring that all data items represented within said array have the same type as one another.

9. A method according to any one of claims **5–8**, wherein said array label identifies said same type.

10. A method according to any one of claims **5 8**, wherein said mark-up language is XML, said at least one element is expressed as an XML element, and said array label is expressed as an XML attribute of said XML element such that the dimension n is given by the value of said XML attribute.

11. A method according to any one of claims **1–4** and **5–8**, wherein said message is an XML document.

12. A method according to any one of claims **1–4**, wherein said message includes a type label associated with the nesting element at each said element nesting level and designating said nesting element as having an array type.

13. A method according to claim **12**, wherein said nesting element contains at least one member element representing a data item, with all data items represented by the direct children elements of said nesting element being of the same type as one another.

14. A method according to claim **13**, wherein said message includes an array label associated with said nesting element and said array label indicates the type associated with all data items represented by said direct children elements.

15. A method according to any one of claims **1–4**, wherein said message includes an array label associated with said nesting element and said array label indicates a value of n but does not indicate a size for each of said n dimensions.

16. A method according to any one of claims **1–4**, wherein said message includes an array label associated with said at least one element and requiring that all data items contained within said array have the same type as one another.

17. A method according to claim **16**, wherein said array label identifies said same type.

18. A method according to any one of claims **1–4**, wherein all elements in said message designating data items are associated with type labels.

19. A method of invoking a service at a first machine from a second machine, comprising the steps of:

generating a service invocation request message at said second machine in compliance with a markup language-based message encoding, wherein said message includes plural elements representing data items of at least one argument and associated with type labels selected from an encoding group having a predetermined number of members, including at least a first type label for designating an element containing lexical data, and a second type label for designating an element containing other elements associated with type labels selected from said group, and a third type label for designating an element containing other elements associated with type labels selected from said group; and

transmitting said message.

20. A method according to claim **19**, wherein said group includes at least a fourth type label and a fifth type label always designating an element containing a data item specifying an ID value, wherein said message associa

element having said fourth type label with an ID value, and wherein said message includes an element associated with said fifth type label which specifies said ID value.

21. A method according to claim **19**, wherein said group includes at least one placeholder type label that designates a placeholder element which represents the absence of data.

22. A method of invoking a service at a first machine, comprising the steps of:

receiving at said first machine a service invocation request message generated at a second machine in compliance with a markup language-based message encoding, wherein said message includes plural elements representing data items of at least one argument and associated with type labels selected from an encoding group having a predetermined number of members, including at least a first type label for designating an element containing lexical data, and a second type label for designating an element containing other elements associated with type labels selected from said group, and a third type label for designating an element containing other elements associated with type labels selected from said group; and

invoking said service in response to said message.

23. A method according to claim **22**, wherein said group includes at least a fourth type label and a fifth type label always designating an element containing a data item specifying an ID value, wherein said message associates an element having said fourth type label with an ID value, and wherein said message includes an element associated with said fifth type label which specifies said ID value.

24. A method according to claim **22**, wherein said group includes at least one placeholder type label that designates a placeholder element which represents the absence of data.

25. A method of invoking a service at a first machine, said method comprising the steps of:

receiving at said first machine a service invocation request;

invoking said service in response to said request; and

transmitting from said first machine a service invocation reply message in compliance with a markup language-based message encoding, wherein said message includes plural elements representing data items of at least one argument and associated with type labels selected from an encoding group having a predetermined number of members, including at least a first type label for designating an element containing lexical data, and a second type label for designating an element containing other elements associated with type labels selected from said group, and a third type label for designating an element containing other elements associated with type labels selected from said group.

26. A method according to claim **25**, wherein said group includes at least a fourth type label and a fifth type label always designating an element containing a data item specifying an ID value, wherein said message associates an element having said fourth type label with an ID value, and wherein said message includes an element associated with said fifth type label which specifies said ID value.

27. A method according to claim **25**, wherein said group includes at least one placeholder type label that designates a placeholder element which represents the absence of data.

28. A method of invoking a service at a first machine, said method comprising the steps of:

transmitting a service invocation request from a second machine; and

receiving at said second machine a service invocation reply message in compliance with a markup language-based message encoding, wherein said message includes plural elements representing data items of at least one argument and associated with type labels selected from an encoding group having a predetermined number of members, including at least a first type label for designating an element containing lexical data, and a second type label for designating an element containing other elements associated with type labels selected from said group, and a third type label for designating an element containing other elements associated with type labels selected from said group.

29. A method according to claim **28**, wherein said group includes at least a fourth type label and a fifth type label always designating an element containing a data item specifying an ID value, wherein said message associates an element associated with said fourth type label with an ID value, and wherein said message includes an element associated with said fifth type label which specifies said ID value.

30. A method according to claim **28**, wherein said group includes at least one placeholder type label that designates a placeholder element which represents the absence of data.

31. A method of invoking a service at a first machine from a second machine, comprising the steps of:

generating a service invocation request message at said second machine in compliance with a markup language-based message encoding, wherein said message includes plural elements representing data items of at least one argument and associated with type labels selected from an encoding group having a predetermined number of members, including at least a first type label for designating an element containing lexical data, and a second type label for designating an element containing other elements associated with type labels selected from said group, said encoding providing a lexical type indicator associated with an element having said first type label and wherein an element associated with said first type label with no lexical type indicator is assumed to contain data of string lexical type; and

transmitting said message.

32. A method of invoking a service at a first machine, comprising the steps of:

receiving at said first machine a service invocation request message generated at a second machine in compliance with a markup language-based message encoding, wherein said message includes plural elements representing data items of at least one argument and associated with type labels selected from an encoding group having a predetermined number of members, including at least a first type label for designating an element containing lexical data, and a second type label for designating an element containing other elements associated with type labels selected from said group, said encoding providing a lexical type indicator associated with an element having said first type label and wherein an element associated with said first type label with no lexical type indicator is assumed to contain data of string lexical type; and

invoking said service in response to said message.

33. A method of invoking a service at a first machine, said method comprising the steps of:

receiving at said first machine a service invocation request;

invoking said service in response to said request; and

transmitting from said first machine a service invocation reply message in compliance with a markup language-

R-847

based message encoding, wherein said message includes plural elements representing data items of at least one argument and associated with type labels selected from an encoding group having a predetermined number of members, including at least a first type label for designating an element containing lexical data, and a second type label for designating an element containing other elements associated with type labels selected from said group, said encoding providing a lexical typo indicator associated with an element having said first type label and wherein an element associated with said first type label with no lexical type indicator is assumed to contain data of string lexical type.

34. A method of invoking a service at a first machine, said method comprising the steps of:

transmitting a service invocation request from a second machine; and

receiving at said second machine a service invocation reply message in compliance with a markup language-based message encoding, wherein said message includes plural elements representing data items of at least one argument and associated with type labels selected from an encoding group having a predetermined number of members, including at least a first type label for designating an element containing lexical data, and a second type label for designating an element containing other elements associated with type labels selected from said group, said encoding providing a lexical type indicator associated with an element having said first type label and wherein an element associated with said first type label with no lexical type indicator is assumed to contain data of string lexical type.

35. A method according to any one of claims 19–28, wherein said encoding group includes a fourth type label for designating an element containing other elements associated with type labels selected from said group.

36. A method according to any one of claims 19–28, wherein said encoding group includes a fourth type label for designating an element uniquely identifying another element within a particular message.

37. A method according to claim 36, wherein said encoding group includes a fifth type label for designating the absence of data.

38. A method according to claim 37, wherein said encoding group includes a sixth type label for designating an element containing other elements associated with type labels selected from said group.

39. A method according to any one of claims 19–28, wherein said encoding group includes a fourth type label for designating the absence of data.

40. A method according to any one of claims 19, 22, 25, 28, wherein said third type label designates an element containing an n-dimensional array (where n is an integer such that $n \geqq 1$) of elements associated with type labels selected from said encoding group.

41. A method according to any one of claims 19, 22, 25, 28, wherein said encoding provides a lexical type indicator associated with an element having said first type label.

42. A method according to claim 41, wherein an element associated with said first type label with no lexical type indicator is assumed to contain data of string lexical type.

43. A method according to claim 41, wherein said markup language is XML, said elements are expressed as XML elements, said type labels are expressed as XML element type names, and said lexical type indicator is expressed as an XML attribute on an XML element associated with said first type label, with the lexical type of the data contained in said XML element being designated by the value of said XML attribute.

44. A method according to any one of claims 19, 22, 25, 28, wherein said encoding group further includes a fourth type label for designating an element representing a numeric value.

45. A method according to any one of claims 19, 22, 25, 28, wherein said encoding group includes multiple type labels each designating a respective different type of lexical data contained in an associated element.

46. A method according to any one of claims 19, 22, 25, 28, wherein said message further includes a semantic label for at least one data item of an argument in said message.

47. A method according to claim 46, wherein said markup language is XML, said at least one element is expressed as an XML element, and said semantic label is expressed as the value of an XML attribute on said XML element.

48. A method according to any one of claims 21, 24, 27, 30, wherein said placeholder element represents a programming language null object reference.

49. A method according to any one of claims 21, 24, 27, 30, wherein said placeholder element identifies an element contained elsewhere in said message.

50. A method according to any one of claims 21, 24, 27, 30, wherein said message includes a second type label associated with said placeholder element.

51. A method according to claim 50, wherein said message includes a semantic label associated with said placeholder element.

52. A method according to any one of claims 21, 24, 27, 30, wherein said message includes a semantic label associated with said placeholder element.

53. A method according to any one of claims 20, 23, 26, 29, wherein said encoding permits any element in a message to be associated with an ID which uniquely identifies said element within said message.

54. A method according to claim 53, wherein said markup language is XML, said element is expressed as an XML element, and said ID is associated with said element via an XML attribute on said XML element whose value is said ID.

55. A method according to any one of claims 5–8 or 20, 23, 26, 29, 21, 24, 27, 30, wherein all elements of said message representing data items are associated with type labels.

* * * * *

Exhibit V. Glushko and McGrath, Document Engineering: Analyzing and Designing Documents for Business Informatics and Web Services (MIT Press 2005), 4 pp. with cover.

DOCUMENT ENGINEERING

ANALYZING AND DESIGNING DOCUMENTS FOR BUSINESS INFORMATICS & WEB SERVICES

Robert J. Glushko and Tim McGrath

The MIT Press
Cambridge, Massachusetts
London, England

This book was set in Bauer Bodoni and Eurostile by Andrea R. Nelson.
Printed and bound in the United States of America.

Library of Congress Cataloging-in-Publication Data

Glushko, Robert J.
Document engineering: analyzing and designing documents for business informatics and Web services / Robert J. Glushko and Tim McGrath.
p. cm.
Includes bibliographical references and index.
ISBN0-262-07261-0 (alk. paper)
1. Text processing (Computer science). 2. Electronic data interchange.
3. Commercial documents—Data processing. I. McGrath, Tim. II. Title.

QA76.9.T48 G54 2005 2001030659
005—dc22

when new documents must be developed for new business processes. But like mainframes and fax machines, EDI can still claim "I'm Not Dead Yet."[51]

4.3.1.2 XMLification

When XML emerged in the late 1990s as the preferred syntax for describing document formats, the EDI standards began to "XMLify," and scores of XML business vocabularies emerged.[52] As with early efforts in EDI, most of the latter were developed in specific vertical industries by trade associations or industry consortia to reduce the development and integration costs for small and medium-sized enterprises that could not afford to invest in EDI solutions.

New XML specifications often reinvent definitions of common information components

But while each new XML specification for a particular industry was a step forward for that industry, they have proliferated definitions of information components that cut across different industries. Each vocabulary reinvented descriptions of businesses and individuals, measurements, date and time, location, country codes, currencies, business classification codes, and basic business documents like catalogs, purchase orders, and invoices. As is often the case with new technologies, it was two steps forward and one step back.

The earliest effort to attack the problem of semantic overlap among XML vocabularies for business applications was the XML Common Business Library, whose first version was released in 1997. XCBL was a freely distributed set of XML business documents and a set of reusable components common to many business processes. XCBL, like many models of business information, is tied to specific technologies or syntaxes such as XML schemas. We call them document implementation models. This means that they are typically published as libraries of XML schemas with the expectation that they will be reused at this physical level. The underlying concepts and meanings encoded in the vocabularies are only implicit or, at best, incompletely documented.

Because of the physical level of the models, syntax differences like those between X12 and UN/EDIFACT with EDI, or between either of these and an XML vocabulary, can get in the way of doing electronic business, even if the concepts underlying the documents being exchanged are compatible. Communication usually requires a knowledgeable person to manually create a semantic map between corresponding syntactic components in the pair of models. This has given rise to a category of integration technology that attempts to reuse these semantic maps.[53]

The reason physical level mapping is difficult is that it requires a common abstract view that defines the concepts involved rather than the implementation technology. So we need conceptual counterparts to our various physical models (see Section 4.3.2).

4.3.1.3 Information Aggregations

Information aggregations occur where documents or data from numerous sources are brought together to create a consolidated information resource that is more valuable than the sum of the sources. In business informatics this composite resource is typically called a data warehouse or data mart. Another common composite pattern is a multivendor catalog that includes product information from many manufacturers or suppliers. More examples can be seen in documents such as daily shipping schedules and stock market trading tables.

Composite information sources can be created by extracting and transforming the original information and are usually built during "off hours" to minimize the impact on production systems, but as businesses become more global it is always "on hours" somewhere. So the challenge facing the enterprise to keep the composite repository accurate becomes more difficult as the source information becomes more volatile.

An alternative approach is to create a virtual repository or virtual catalog in which the metadata from each source is aggregated into the composite resource, not the content itself. This composite metadata enables the content information to be referenced from its source and dynamically transformed to the target implementation model when the information is requested.[54]

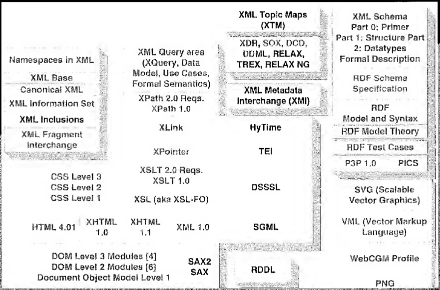
Another information composition pattern is syndication, the consolidation and distribution of information products. This is widespread in traditional publishing with

R-853

Exhibit W. Sall, Kenneth B., XML Family of Specifications: A Practical Guide (Addison Wesley 2002), 1 pp. with cover.

XML Family of Specifications

A Practical Guide



Volume 2

Includes Contributions from Ora Lassila on RDF and G. Ken Holman on XSLFO

Kenneth B. Sall

XML FAMILY OF SPECIFICATIONS

A PRACTICAL GUIDE

Kenneth B. Sall

Addison-Wesley

Boston • San Francisco • New York • Toronto • Montreal
London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

The publisher offers discounts on this book when ordered in quantity for special sales. For more information, please contact:

Pearson Education Corporate Sales Division
201 W. 103rd Street
Indianapolis, IN 46290
(800) 428-5331 corpsales@pearsoned.com
Visit AW on the Web: *www.awprofessional.com*

Library of Congress Cataloging-in-Publication Data

Sall, Kenneth B.
XML family of specifications : a practical guide / Kenneth B. Sall.
p. cm.
Includes bibliographical references and index.
ISBN 0-201-70359-9 (alk. paper)
1. XML (Document markup language) 2. Web sites—Design. 3. Internet programming.
I. Title
QA76.76.H94 S22 2002
005.7'2--dc21 2002022640

ISBN 0-201-70359-9
Text printed on recycled paper

1 2 3 4 5 6 7 8 9 10–CRS–0605040302
First Printing, May 2002

Organization or Initiative	*Brief Description from Web Site*
UCC	*Uniform Code Council* states that "[t]he goal of the [XML] program is to execute XML Strategy that will result in implementation of the XML standards for Business-to-Business (B2B) and Business-to-Consumer (B2C) electronic commerce." See *http://www.uc-council.org/.*
UDDI	*Universal Discovery Description and Integration* initiative, according to the UDDI Executive White Paper (Sept. 2000), "creates a global, platform-independent, open framework to enable businesses to (1) discover each other, (2) define how they interact over the Internet, and (3) share information in a global registry that will more rapidly accelerate the global adoption of B2B eCommerce." Based on XML, DNS, HTTP, and SOAP. See *http://www.uddi.org/* and *http://www.uddicentral.com/.*
WSDL	*Web Services Description Language* is the XML vocabulary that describes services and service providers (enabling them to be located using UDDI). See *http://xml.coverpages.org/wsdl.html.*
xCBL	Commerce One's *XML Common Business Library* promotes "cross-industry exchange of business documents such as product descriptions, purchase orders, invoices, and shipping schedules." Another goal is "to make the business documents, forms, and messages that flow between businesses comprehensible to each business no matter what computer system is used." The insightful CBL effort predates the XML 1.0 Recommendation, dating back to 1997. xCBL uses a mature schema specification (*SOX*) which is a forerunner of the W3C's XML Schema standard. See *http://www.xcbl.org/* for details.
XML.org and its parent organization, OASIS	Diverse membership and participation from many industries and businesses. Maintains an extensive DTD/Schema catalog, and repository. "The XML Catalog lists organizations known to be producing industry-specific or cross-industry XML Specifications. The XML Catalog is intended to evolve into the XML.ORG Registry & Repository for XML Schemas, DTDs and specifications." Catalog categories include accounting, advertising, astronomy and space, automotive, banking, communication, computer graphics, content syndication, customer relationship management, directory services, distributed management, economics, education, electronic commerce, electronic data interchange, enterprise resource planning (ERP), financial and capital, forms, healthcare, human resources, industrial automation, intellectual property rights, insurance, legal, music, news, publishing, real estate, retail, science (chemistry, biology, etc.), software, supply chain management (SCM), travel, user interface, voice, Web applications, workflow, among many others. Sponsored creation of comprehensive XML and XSL conformance suites. Took over the xml-dev developer's mailing list in early 2000. Parent organization, Organization for the Advancement of Structured Information Standards (OASIS), has been a key player in the SGML community. "OASIS is a member consortium dedicated to the advancement of structured information standards, such as SGML, XML and CGM." See *http://www.xml.org/* and *http://www.oasis-open.org/.*
XML/EDI Group	Activists since 1997 with large mailing list of interested parties; promotes repository concept; introduced XLink-related BizCodes. See *http://www.geocities.com/WallStreet/Floor/5815/* and *http://www.bizcodes.org/.*